UNIVERSITY of CALIFORNIA

Santa Barbara

Analyzing and Processing Big Real Graphs

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Xiaohan Zhao

Committee in Charge:

Professor Ben Y. Zhao, Chair

Professor Haitao Zheng

Professor Xifeng Yan

December 2014

The Dissertation of Xiaohan Zhao is approved.

Professor Haitao Zheng

Professor Xifeng Yan

Professor Ben Y. Zhao, Committee Chair

December 2014

Analyzing and Processing Big Real Graphs

Xiaohan Zhao

To my parents and my husband, for your endless love and unconditional support that make me happy everyday and lead me to where I am.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor, Prof. Ben Y. Zhao. I am amazingly fortunate to have an advisor who gives me the invaluable guidance to my research, and grants me great freedom to pursue independent work. In the last five years, Ben has taught me how to conduct good research with his patience, enthusiasm and immense knowledge. Meanwhile, he is also my advisor outside of academia. He always shares with me his wisdom on how to live a happy life, laughs with me about funny jokes, and has trained me to be a fan of X-men. As I always say, Ben is my academic "father", who always gives me great support and encouragement.

Second, I would like to gratefully and sincerely thank Prof. Haitao Zheng. She is like my second advisor. Through our collaboration, I have learned to be careful with each step and every detail of my research, which is critical to my future career. I am deeply grateful for her insightful advice on my work, talks and papers.

I also sincerely thank my committee member Prof. Xifeng Yan. I am deeply grateful for his valuable feedback and insightful suggestions on my work.

My sincere thanks go to Dr. Alessandra Sala and Prof. Christo Wilson. It is my pleasure to work with them on multiple projects. I am also grateful to Prof. Xia Zhou, Dr. Lei Yang, Dr. Lili Cao, Dr. Krishna Puttaswamy, and Dr. Vinod Kone for their various forms of help during my graduate study. Also I appreciate the efforts of Adelbert Chang who worked with me on the incredible VLDB paper.

I am indebted to all the current members of SAND Lab: Zengbin Zhang, Gang Wang, Yibo Zhu, Ana Nika, Qingyun Liu, Lin Zhou, Tianyi Wang, and

Bolun Wang. Thank them for helping me through hard times and sharing happy moments with me.

I would like to acknowledge my collaborators and mentors for their valuable advice and numerous discussions: Prof. Sabrina Gaito, Dr. Atish Das Sarma, Dr. Walter Willinger, Xiao Wang, Dr. Antony Rowstron, and Chris Conrad.

Finally, and most importantly, I would like to thank my loving family. Thanks to my father Wenjie Zhao and my mother Chunyan Zhou for their endless love, and for always being there for me. Thanks to my husband Bo Zong for his unconditional love, support and encouragement. Thank you.

Curriculum Vitæ

Xiaohan Zhao

Education

2014	Ph.D in Computer Science, University of California, Santa Barbara.
2009	Master of Science in Electronic Engineering, Tsinghua University, China.
2007	Bachelor of Enginnering in Electronic Engineering, Tsinghua University, China.

Experience

08/2009 – 08/2014	Research Assistant, University of California, Santa Barbara.
06/2013 – 09/2013	Research Intern, Microsoft Research, Cambridge, UK.
10/2012 – 12/2012	Research Intern, Microsoft Research, Cambridge, UK.
06/2012 – 09/2012	Software Intern, LinkedIn.

Publication

Matteo Zignani, Sabrina Gaito, Gian Paolo Rossi, **Xiaohan Zhao**, Haitao Zheng, and Ben Y. Zhao. "Link and Triadic Closure Delay: Temporal Metrics for Social Network Dynamics." *International Conference on Weblogs and Social Media (ICWSM)*, Jun. 2014.

Zengbin Zhang, Lin Zhou, **Xiaohan Zhao**, Gang Wang, Yu Su, Miriam Metzger, Haitao Zheng, and Ben Y. Zhao. "On the Validity of Geosocial Mobility Traces." *ACM Workshop on Hot Topics in Networks (HotNets)*, Nov. 2013.

Xiaohan Zhao, Adelbert Chang,Atish Das Sarma, Haitao Zheng, and Ben Y. Zhao. "On the Embeddability of Random Walk Distances." *Very Large Data Bases (VLDB)*, Sept. 2013.

Xiaohan Zhao, Alessandra Sala, Christo Wilson, Xiao Wang, Sabrina Gaito, Haitao Zheng, and Ben Y. Zhao. "Multi-scale Dynamics in a Massive Online Social Network." *Internet Measurement Conference (IMC)*, Nov. 2012.

Sabrina Gaito, Matteo Zignani, Gian Paolo Rossi, Alessandra Sala, **Xiaohan Zhao**, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. "On the Bursty Evolution of Online Social Networks." *ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research (HotSocial)*, Aug. 2012.

Gang Wang, Christo Wilson, **Xiaohan Zhao**, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y. Zhao: "Serf and Turf: Crowdturfing for Fun and Profit." *International World Wide Web Conference (WWW)*, Apr. 2012.

Xiaohan Zhao, Alessandra Sala, Haito Zheng, and Ben Y. Zhao: "Efficient Shortest Paths on Massive Social Graphs." *IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, Oct. 2011. (**Invited Paper**)

Alessandra Sala, **Xiaohan Zhao**, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. "Sharing Graphs using Differentially Private Graph Models." *Internet Measurement Conference (IMC)*, Nov. 2011.

Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. "Orion: Shortest Path Estimation for Large Social Graphs." *Workshop on Online Social Networks (WOSN)*, Jun. 2010.

Xiaohan Zhao, Xiaoxiao Song, Xiao Wang, Yang Chen, Beixing Deng, and Xing Li. "Analysis of Security Policy in Practical Internet Coordinates." *International Journal of Security and Its Applications*, Vol. 3, No. 1, 2009.

Yang Chen, Xiao Wang, Xiaoxiao Song, Eng Lua, Cong Shi, **Xiaohan Zhao**, Beixing Deng, and Xing Li."Phoenix: Towards an Accurate, Practical and Decentralized Network Coordinate System." *International IFIP-TC 6 Networking Conference (NETWORKING)*, May 2009.

Xiaoxiao Song, **Xiaohan Zhao**, Eng Keong Lua, Zengbin Zhang, Beixing Deng, and Xing Li. "SLINCS: A Social Link based Evaluation System for Network Coordinate Systems." *IEEE Consumer Communications & Networking Conference (CCNC)*, Jan. 2009. (**Short Paper**)

Xiaohan Zhao, Xiaoxiao Song, Xiao Wang, Yang Chen, Beixing Deng, and Xing Li. "Attacks against Network Coordinate System: Vulnerable PIC." *International Symposium on Computer Science and its Applications (CSA)*, Oct. 2008.

Abstract

Analyzing and Processing Big Real Graphs

by

Xiaohan Zhao

As fundamental abstractions of network structures, graphs are everywhere, ranging from biological protein interaction networks and Internet routing networks, to emerging online social networks. Studying graphs is critical to understanding the fundamental processes behind the networks, and of practical importance in experimental research. Although many studies on graphs have been carried out in decades, most of the work focused on small or synthetic graphs. In recent years, because of the unprecedented increase of existing networks and the emergence of new complex networks, more and more big real graphs are becoming available. Compared to the graphs studied in prior work, the graphs from these networks are significantly different in scale, level of dynamics and structure.

In this dissertation, we tackle three important graph research problems caused by the significant differences of the big real graphs: efficient node distance computation, graph dynamic analysis and modeling, and graph privacy.

First, we target on a fundamental graph analysis problem, *i.e.* node distance computation. As a primitive of graph analysis and network applications, the computation of shortest path or random walk distances is computationally expensive, and difficult to scale with the sheer size of big real graphs. To address the scalability issue, we design a novel node distance computation method, named *graph coordinate systems*, to efficiently estimate node distances with high accuracy.

Our second work is to understand and model the dynamic processes in big real graphs. Specifically, we propose methods to analyze graph dynamics at multiple network scales and explore temporal properties of network growth. Through measurements on Renren first two-year dynamic data, we find independent and predictable processes at different network levels, and detect self-similar properties in its edge creation process. Based on the observations, we propose a new dynamic graph model to capture both temporal and spatial properties. Calibrated with the Renren dataset, our model successfully produces synthetic graphs showing similar dynamic properties.

Finally, to address privacy issue in sharing graphs, we design a graph privacy system to guarantee the required level of privacy. The goal of our work is to design a system that can both maintain a meaningful graph structure and provide strong privacy guarantee. To navigate the tradeoff between the strength of privacy and graph structure utility, we propose a differentially-private graph model. Our rigorous proof shows that the graphs produced by the system can achieve the required level of privacy. By running the system on real graphs collected from Facebook, Internet, and Web, the results demonstrate that the generated synthetic graphs match the original graphs in terms of graph structural metrics and application-level performance.

In summary, to analyze and process the graphs from today's large complex networks, we work on three important problems, including efficiently computing node distances in massive graphs, analyzing and modeling high volume of dynamics in big real graphs, and protecting graph privacy in sharing graphs. We propose novel solutions to address these problems. Through our extensive experiments, we show that our designs perform consistently well on big real graphs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are fundamental abstractions of network structures. They are everywhere, ranging from biological protein interaction networks and Internet routing networks, to emerging online social networks. Studying graphs is critical to understanding the fundamental processes behind networks, and of practical implications in real world applications. For example, understanding and modeling influence propagation in online social networks, such as Facebook, Twitter, and Renren, is important for the service providers to successfully launch social advertising [33]. In previous decades, there have been many studies on graphs [104, 102, 182, 117, 5, 6, 18]. Most of the prior work focused on small or synthetic graphs [104, 102, 182, 117, 5, 6, 18], such as arXiv citation networks, DBLP collaboration networks, and Email communication networks.

More recently, because of the unprecedented increase of existing networks, such as Internet, and the emergence of new networks like online social networks, more and more big real graphs are becoming available. Compared to the graphs studied in prior work, these big real graphs are significantly different in the following three aspects. First, big real graphs are much larger in scale. For example, in online

social networks, like Facebook, Twitter, and LinkedIn, there are millions or even billions of users while only hundreds of thousands of nodes exist in the graphs in prior studies, such as arXiv networks and DBLP networks [104, 102, 182, 117, 5, 6, 18]. Second, big real graphs have higher level of dynamics. Compared with online social networks like Renren, which has hundreds of thousands of new users and more than 1 millions new edges *per day*, there are only thousands of new nodes and new edges *per year* in the arXiv network [104]. Third, from the perspective of graph structure, big real graphs have significantly distinct properties. Take graph density as an example. In online social networks, such as Facebook, its average degree of each user are more than 190 [12], which is several times larger than the average degree in the prior graphs [104].

Thus, it is challenging to understand big real graphs. First, because of their sheer size, many traditional graph algorithms are not scalable in big real graphs, which makes processing these graphs extremely difficult. For example, to compute one shortest path in a graph with millions of nodes and billions of edges, it can take up to minutes or even an hour using breadth-first search (BFS), which is too slow to support any real-time applications based on shortest paths. Second, although many studies have worked on dynamic graphs [104, 102, 5, 6], few progress is reported on understanding high volume of graph dynamics. In order to understand dynamic processes in big real graphs, systematic analysis methods and comprehensive dynamic graph models are desired. Third, protecting privacy in sharing big real graphs is a new issue. This is because the structure of big real graphs may contain sensitive information, such as strength of social ties, number and frequency of social interactions, and information flows in online social networks. Sharing such graphs raises risks to expose users' private information to

the public. Therefore, how to preserve graph privacy is becoming one important problem in studying big real graphs.

My research work aims to address the above three challenges in big real graphs, and the statement of this dissertation is as follows:

To analyze and process an increasing number of big real graph datasets today, we need to build tools and algorithms to address issues of scale, dynamics, and data privacy.

Driven by the statement, the following are the three goals in this dissertation. First, we target on a fundamental graph processing problem, *i.e.* node distance computation. As a primitive of graph analysis and network applications, the computation of shortest path or random walk distances is computationally expensive. In this dissertation, we design a new node distance computation method, named *graph coordinate systems*, which can accurately approximate node distances in constant time. The second goal of the dissertation is to understand and model dynamic processes in big real graphs. Specifically, we propose methods to analyze graph dynamics at multiple network scales, and explore temporal properties of network growth. Based on the observations, we design a new dynamic graph model to capture both temporal and spatial dynamic properties. Finally, we tackle privacy issue in sharing graphs. To protect graph privacy, we design a differentially-private graph model, which can guarantee the required level of privacy while maintaining a meaningful graph structure. In the following, we briefly introduce the work included in this dissertation.

1.1 Efficient Node Distance Computation

Node distance computation, including shortest path distances and random walk distances, is computationally expensive, and difficult to scale to big real graphs. To address this problem, we propose an efficient approach, named *graph coordinate systems*, which can accurately estimate node distances in real time.

At high level, in graph coordinate systems, we embed graph nodes to a geometric space. After the embedding, each node can be represented by its geometric coordinates. With the coordinates, we estimate the distance between any pair of nodes by computing their geometric distance, which is a constant time computation. Based on this idea, we propose the framework of graph coordinate systems, which is a landmark-based scheme.

To accurately embed shortest path distance in large graphs, we explore two key design decisions in the implementation of a graph coordinate system, including choice of geometric spaces and scalable embedding process. By implementing three graph coordinate systems using three well-studied geometric spaces, such as Euclidean space, spherical space, and hyperbolic space, we study the impact of geometric spaces on estimation accuracy, and find that the hyperbolic space performs the best out of the three spaces. Thus, we adopt the hyperbolic space in the design of the graph coordinate system to embed shortest path distances. Further, to scalably embed big real graphs, we naturally parallelize the embedding process across multiple servers. The resulting parallel hyperbolic graph coordinate system is called *Rigel*. Moreover, we propose a heuristic method to locate shortest paths using the generated coordinates. By running the system on different big

real graphs, we show that Rigel can accurately approximate graph shortest path distances in microseconds.

Different from shortest path distances, graph coordinate systems using traditional geometric spaces can not accurately approximate asymmetric random walk distances. To address this challenge, we study the cause of the asymmetry in random walks, and propose a new space with two independent height vectors to explicitly account for the asymmetry. By embedding various large graphs, we show that the new space not only accurately estimates asymmetric distances, such as hitting time and personalized PageRank (PPR), but also improves the accuracy of symmetric distance prediction, like commute time.

1.2 Dynamic Graph Analysis and Modeling

In this work, we analyze and model the high level of dynamics in big real graphs. More specifically, we aim to design systematic dynamic graph analysis methods, and build a complete dynamic graph model that can capture both temporal and spatial dynamic properties. To make it concrete, we focus on analyzing dynamics in a large time-stamped social network, Renren, with 19 million nodes and 199 million edges.

To comprehensively understand dynamics in the Renren network, we measure the network in two dimensions: spatial dimension and temporal dimension. In spatial analysis, instead of considering a single dynamic process in the graph, we understand users' activities at different network levels. In particular, we propose a multi-scale dynamic measurement method, including individual node level, community level, and network level. At each level, we seek for the evidence of the

underlying processes, and learn how they impact users' behavior. Along the way, we also make a number of intriguing observations about dynamic processes in network communities and network-wide events. In temporal analysis, we explore the efforts to detect and identify the existence of self-similarity in Renren's network growth. Self-similarity refers to that the relative variance or volatility of a dynamic process stays similar across different time scales. Because of the non-stationary diurnal pattern in Renren's network growth, it is challenging to detect self-similar properties in a statistically rigorous manner. To overcome the challenge, we use a range of different detection algorithms to reliably identify self-similar properties.

Finally, to capture the observed spatial and temporal properties, we propose a new dynamic graph model. It includes a temporal component and a spatial component, which explicitly accounts for "when" and "where" an edge is created. As a whole, this model can produce a sequence of graph events that captures the evolutionary dynamics in graph structure. By calibrating the model with the Renren network, the generated graphs not only reproduce the self-similar edge creation process but also match the evolution of several structural metrics.

1.3 Graph Privacy

Successes of studies on big real graphs strongly depend on the availability of the graphs from the real networks. However, sharing graph data raises risks to expose sensitive users' data to the public. Unfortunately, current studies [71, 111] only focus on defending a specific attack, and have been proved vulnerable [13, 126, 127]. In our work, our goal is to design a graph privacy system that can

both maintain a meaningful graph structure and provide strong privacy guarantee without any assumptions about attacks.

To balance such tradeoff, we propose a differentially-private graph model, called *Pygmalion*. Given a graph and a desired level of privacy guarantee, the system extracts the graph structure represented by the joint degree distribution, adds a controlled level of noise, and produces synthetic graphs similar to the original graph.

However, by running this method on big real graphs collected from various networks, we find that directly adding the required noise to the graph structure representation introduces high distortion into the generated synthetic graphs. To maintain the utility of differentially private graphs, we further develop a partitioning method, which significantly reduces amount of added noise while providing the same level of privacy guarantee. We run the improved system on the same set of graphs. The results show that the generated synthetic graphs consistently match their original graphs in terms of graph structural metrics and application-level performance.

1.4 Contributions

In this dissertation, there are two key contributions to the study of big real graphs. First, we design novel systems or solutions to address the important problems in big real graph study. In Chapter 2, we propose the framework of graph coordinate systems, which can efficiently approximate node distances with high accuracy. Based on this framework, we implement a practical graph coordinate system to embed shortest path distances. Furthermore, to accurately

capture asymmetric distances, we design a novel embedding space with two independent heights. In Chapter 3, we develop a multi-scale measurement method to understand graph structural dynamics. Also, inspired by lessons from network traffic modeling, we explore the self-similar properties in Renren edge creation process. Based on the measurement results, we propose a new dynamic graph model, which not only captures the structural evolution but also produces the sequence of edge creation events in absolute time. At the last of this dissertation (Chapter 4), we apply differential privacy to address graph privacy issue. The proposed differentially-private graph model can generate synthetic graphs similar to the original graph in terms of structure with a desired privacy guarantee.

Second, we validate our solutions on a range of big real graphs. To demonstrate the generality of our designed systems, including the graph coordinate systems and the differentially-private graph model, we evaluate their performance on a number of big real graphs collected from various networks, including different online social networks and computer networks. The results from these graphs show that in most of big real graphs, our systems can consistently perform well. Our work in Chapter 3 is the first dynamic study on such massive scale. By fitting the model with the Renren dynamic dataset, we generate synthetic graphs showing dynamic properties similar to the Renren network.

1.5 Thesis organization

The roadmap of the dissertation is as follows:

In Chapter 2, we describe the design of graph coordinate systems. We begin with description of the framework of graph coordinate systems. Then we

implementation a graph coordinate system to embed shortest path distances by exploring studying two design decisions, such as embedding space and scalable embedding process. We also run it on three applications to demonstrate its accuracy, and propose a heuristic method to locate shortest path with the generated coordinates. Finally, we explore a new embedding space to explicitly account for the asymmetry of random walk distances, and evaluate its performance on various networks.

In Chapter 3, we elaborate our analysis on the Renren dynamic dataset, and describe a new dynamic model based on the measurement. After describing the dataset and showing basic measurement results, we introduce our measurement on Renren graph structure at different network levels, including nodes, communities, and networks. Second, we apply three methods to detect and identify the self-similar properties in edge creation process, which is an important temporal property in network growth. At last, to reproduce the observed spatial and temporal properties, we propose a new dynamic graph model combining a temporal component and a spatial component, and validate it using the Renren dataset.

In Chapter 4, we propose a differentially-private graph model to protect graph privacy. Following the background, we propose our basic solution by applying differential privacy to graphs. Because of high distortion in the graphs generated by the basic design, we improve the system using a partitioning method, and provide a rigorous proof to show the improved system maintains the same level of differential privacy with much less noise. Finally, we use different big real graphs to evaluate the structure utility of the generated graphs.

At the end of this dissertation, we summarize the work, and discuss future directions.

Chapter 2

Efficient Node Distance Computation

2.1 Introduction

[1] Analysis of large graphs is critical to understanding the ongoing growth of complex networks, such as online social networks, biological protein interaction networks, and Internet router backbone. One important measure in such analysis is to compute node distance. Such distance can be quantified either by shortest path, or by random-walk distances, such as commute time, hitting time and personalized PageRank (PPR).

Computing the shortest path distance between two nodes is a primitive that lies at the core of both graph analysis and complex network applications. For example, in a network of n nodes, computing exact values for node separation metrics like graph radius, diameter and average path length, requires comput-

[1] Abbreviated version of content in this chapter can be found in papers "Orion: Shortest Path Estimation for Large Social Graphs" [186], "Efficient Shortest Paths on Massive Social Graphs" [187], "Fast and Scalable Analysis of Massive Social Graphs" [188], and "On the Embeddability of Random Walk Distances" [184].

ing $O(n^2)$ shortest path. Node distance is also the determining factor for other common graph problems, such as centrality and mutual friend detection.

Unfortunately, current algorithms to compute shortest path distances cannot scale with graph size. For a graph with n nodes and m edges, efficient implementations of traditional algorithms, such as breadth-first-search (BFS), Dijkstra and Floy-Warshall, compute short path between two nodes in $O(n \log n + m)$ time, and all pair shortest-paths in $\Theta(n^3)$ [36]. Tolerable for small graphs, the computation for a single pair of nodes on a large million-node graph can take up to minutes on modern computers [141]. Given the prohibitively high costs of storing precomputed distances, researchers have little choice but to sample portions of the graph or seek approximate results.

In this chapter, we propose a novel method to approximate shortest path distance computation, called as *Graph Coordinate Systems*. A graph coordinate system maps nodes in high dimensional graphs to positions in a geometric space. Using the coordinates associated with each graph node, we can use a simple geometric distance computation to estimate, in constant time, its distance to any other node in the graph.

Moreover, compared to shortest path distances, random-walk distances are more useful in term of quantifying similarity between nodes in graphs. For example, social networks like LinkedIn often provide a measure of similarity between users. Pure shortest path distance cannot reflect the strength of ties between users. An alternative uses the number of paths between nodes. But this fails to capture the impact of a user's degree, *i.e.* m paths between user A and user B is more significant when A and B each have few friends. On the contrary, random walks is a powerful measure of similarity by combining two well studied notions of

affinity between graph nodes, namely the node distance and the number of paths. Intuitively, two nodes are more similar if they are close in terms of their graph distance. Independently, two nodes are more similar if they have more paths between them. Random walk distances, such as commute time, hitting time and PPR, successfully capture both of the notions through the simple iterative random walk process.

However, the computation of random walk distances is also computationally expensive. For example, hitting time is the expected number of random walk hops from a source node to a destination node. Computing the expected hitting time from node A to node B requires computing hundreds of thousands of random walks. Such costs are intractable in today's massive graphs with millions of nodes and billions of edges. Assuming the availability of sufcient memory resources, Computing a single hitting time on a massive graph can take anywhere from minutes to an hour or longer. Thus, it is unsurprising that random walk distances are rarely used in practice.

With the proposed graph coordinate systems, we investigate the possibility of using a geometric space embedding to provide an efficient way to answer queries on random walk distances. Unlike shortest path length, random walk distances, such as hitting time and PPR, are asymmetric, *i.e.* the distance from node A to node B may be not the same as the distance in the reverse direction. In addition, the distances in any geometric space, *e.g.* Euclidean space or hyperbolic space, are symmetric. The asymmetry of random walks can cause high errors in embedding graphs into a geometric space, which is confirmed by our extensive measurements. Based on this observation, we design a new space for graph coordinate systems to account for the asymmetry of random walks.

In this chapter, we make four contributions. First, we propose graph coordinate systems, *i.e.* the use of embedding graphs into geometric spaces, as a new method to approximate node distances in constant time. By explaining the design goals, we describe a landmark-based framework of graph coordinate systems. In the design, we consider and discuss several schemes to select and compute landmarks.

Second, we use graph coordinate systems to embed shortest path distances, and explore the key design decisions in the implementation of graph coordinate systems, including choice of geometric spaces and parallel techniques to fast embed large graphs. The study results in a parallel hyperbolic graph coordinate system named *Rigel*, which estimates shortest path distances in microseconds with high accuracy.

Third, we propose an algorithm to efficiently locate shortest paths between node pairs with the generated coordinates. Comparing with several sketch-based algorithms, our method is more efficient in finding shortest paths, which matches the best accuracy of these algorithms.

Finally, considering the power of random walk distances in applications, we design a new embedding space to capture the asymmetry of random walk distances. Based on the insight of the asymmetry of random walks, we propose two independent heights to capture the intuition of graph density on a per-node basis. In addition, we propose a simple low cost technique to generate ground truth. With the two techniques, we implement a new practical graph coordinate system named *Leo*. By running Leo on a range of big real graphs, the extensive experiments show that using this new graph coordinate system not only accurately

Networks	Node #	Edge #	Avg. Degree
Monterey Bay	6K	31K	10.26
Santa Barbara	26K	226K	17.05
Egypt	246K	1,618K	13.12
Los Angeles	275K	2,115K	15.38
Norway	293K	5,589K	38.15
India	363K	1, 556K	8.57
Flickr	1,715K	15,555K	18.14
Orkut	3,072K	117,185K	76.29
Livejournal	5,189K	48,942K	18.86
Renren	43,197K	1,040,429K	48.17
Collaboration	21,363	91,342	8.55
AS	26,475	533,831	40.33
Citation	34,401	420,828	24.47
P2P	62,562	147,878	4.73
Email	224,832	339,925	3.02
Amazon	262,111	899,792	6.87
Web	325,729	1,117,563	6.86
Planar	265,722	531,441	3.99

Table 2.1: Datasets used in our experiments.

estimates the asymmetric random walk distances, but also significantly improves the accuracy of symmetric distances.

2.2 Experiment Datasets and Evaluation Metrics

Before we describe the details of graph coordinate system designs, we first introduce the graph datasets used in later sections, and explain the metrics used to evaluate the performance of the systems.

2.2.1 Datasets

Throughout this chapter, we use a number of anonymized graph datasets gathered from different networks to evaluate our system design. We utilize 17 graphs

listed in Table 2.1, ranging in size from $6K$ nodes and $31K$ edges, to 43 million nodes and 1 billion edges. We also generate one synthetic planar graph for the evaluation. We use these graphs to demonstrate the scalability and applicability of graph coordinate systems across a variety of graph topologies.

Listed in Table 2.1, six of the graphs, Monterey Bay, Santa Barbara, Egypt, Los Angeles, Norway and India, are Facebook regional networks [179]. We also use four large graphs collected from four different online social networks, *i.e.* Flickr, Orkut, Livejournal [125], and Renren [78]. The next seven graphs in Table 2.1 are from various networks. They are a collaboration network graph from arXiv [105], an Internet Autonomous system (AS) graph from CAIDA [104], a citation graph from arXiv [104], a snapshot of the Gnutella peer-to-peer file sharing network [104], a measurement Email network graph of a large European research institution [105], an Amazon product co-purchasing graph [100], and a web graph from North Dame [7]. Finally, we produce a synthetic planar graph using the Dorogovtsev-Goltsev-Mendes Internet Model [46]. In each section of this chapter, we run the experiment on a subset of the graphs in Table 2.1 to show the performance of the graph coordinate systems.

2.2.2 Evaluation Metrics

Accuracy Metrics. We use *Relative Error* to evaluate the accuracy of graph coordinate systems. For each node pair in a graph, the relative error is the ratio of the absolute difference between the ground truth node distance and the geometric node distance (the estimated distance) to the ground truth distance. A smaller relative error means the estimated node distance matches the ground truth better.

We also use two other metrics to describe the accuracy of the system. The first metric is *average relative error* (ARE) of the estimated distances, which is used in evaluating the accuracy of the graph coordinate systems when embedding shortest path distance. The second accuracy metric is the 90^{th} percentile relative error over all node pairs, called 90^{th} *relative error*, which is the key metric to measure the accuracy of embedding random-walk distances.

Efficiency Metrics. To investigate the efficiency of the system, we use *computation time*. It includes two parts, *i.e.* the system bootstrap time and the response time for per query. First, the system bootstrap time involves two main opertation time, including the time to measure distances between each landmark and all the other nodes and the time to compute coordinates for all nodes. As shown later, since the complexity of embedding scales linearly with graph size, we parallelize the bootstrap process across multiple servers. In this case, the parallel bootstrap time is defined as the longest computation time for the servers used in the paralle embedding process. Second, the response time for per query is measured as the average time to compute pairwise node distances using generated coordinates. Our measurement results show that the per-query response time in our system is almost constant time in microseconds after up to several hours of bootstrap process. This bootstrap time is acceptable since bootstrapping is a one-time cost, which enable us to respond queries in real time.

2.3 Graph Coordinate Systems

The goal of our work is to accurately and efficiently estimate distance between any two nodes in large graphs. To achieve this goal, we propose *graph coordinate systems*, which use a geometric coordinate space to capture node distances on large graphs.

In this section, we first explain the goals of graph coordinate system design. Then, based on the goals, we describe a landmark-based framework of graph coordinate systems, and discuss different methods to select and compute landmarks. Finally, through experiments on several graphs collected from Facebook, we study the impact of three landmark selection schemes on the accuracy.

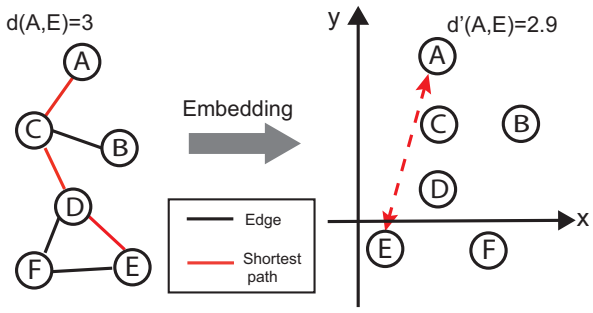


Figure 2.1: An example to map graph into a Euclidean space. The shortest path between node A and E is 3 hops (left) and their Euclidean distance is 2.9 hops (right).

2.3.1 Design Goals

Graph coordinate systems are designed to efficient approximate node shortest path distances with high accuracy. At a high level, this approach captures complex graph structures by embedding node shortest path distance relationships into a

geometric space, such as a Euclidean space, a spherical space and a hyperbolic space. Each node is represented by a set of coordinate in the geometric space such that its distance to another node in the geometric space matches its shortest path length to the node in the actual graph. For example in Figure 2.1, the shortest path between node A and node E is 3 hops in the graph and the Euclidean distance calculated using their coordinates is 2.9.

Graph coordinate systems work in two phases. First, nodes in the graph are iteratively added to the coordinate space, the position of each node being calibrated by ground truth node-distance measurements. This "calibration phase" is where a graph coordinate system incurs its one-time computational overhead. Once all nodes in the graph have been added, the resulting system can be integrated with graph applications to answer node distance queries with estimates.

Since the per-query computation cost is $O(1)$, the focus of our design is to ensure the calibration phase is computationally efficient, and the results are as accurate as possible. More specifically, our goals are three-fold:

- Scalability. The computational cost of the calibration phase must scale linearly with the number of nodes, *i.e.* $O(n)$.
- Accuracy. While individual node distance predictions might incur reasonable errors, predictions should approximate ground truth at the large scale.
- Fast convergence. Impact of individual node calibrations should be localized, *i.e.* should not trigger significant new adjustments to their neighbors.

Based on these goals, we describe the design and explain key decisions in the next section.

2.3.2 A Landmark-based Framework

To accurately translate pairwise hop-count distances in the graph into geometric distances in the coordinate space, the framework of graph coordinate systems is based on landmarks, where the positions of all nodes are calibrated with their relative distances to a fixed number (k) of chosen landmark nodes. Landmark nodes are initially chosen from the entire graph based on their position and degree of connectivity.

We use a landmark-based scheme in the framework for two main reasons. First and foremost, we wish to minimize the number of shortest path computations needed to establish ground truth on the actual graph, since each computation can, in the worst case, require a full traversal of the graph. Using a landmark approach, we limit the total number of Breadth-First-Search operations to k, the number of landmarks. Each BFS computes the shortest path distance from a landmark to all other nodes. Computing BFS for all landmarks essentially precomputes all values needed to calibrate all nodes in the graph.

The second advantage of a landmark-based scheme is that the positions of incoming nodes depend only on the landmark nodes. This bounds the number of operations required to compute a node's position, guaranteeing fast convergence. In contrast, in decentralized models adding a new node will often force its nearby neighbors to make adjustments on their position, a process that can propagate adjustments iteratively throughout the entire space.

Scalable Landmark Coordinates. Intuitively, the number of landmarks used to calibrate a graph should have a direct impact on the accuracy of the Euclidean mapping. The highly connected and complex nature of social graphs leads us to

believe that an accurate graph coordinate system requires a significant number of landmarks. The challenge is to find a way to accurately and quickly compute the coordinates for a large number of landmarks.

To compute a node's D-dimension coordinates, we consider Simplex Downhill algorithm [128] by minimizing the sum of squares of prediction errors. The algorithm runs in $O(k^2 \cdot D)$ time to compute coordinates of k landmarks. Since running Simplex Downhill on our desired number of landmarks (up to 100 in our study) is computationally expensive, we propose a new approach, where we separate our landmarks into two groups, a small initial group of 16 landmarks, and a larger secondary group composed of the remaining landmarks.

We leverage the Simplex Downhill algorithm to compute the coordinates for the initial ($k_I = 16$) landmarks, thus its asymptotical complexity is $O({k_I}^2 \cdot D)$. The secondary group of landmarks calibrate their positions using the initial k_I landmarks as anchors, contributing to a computational complexity of only $O(k_I \cdot D)$ each. Thus, the total time required to compute landmark coordinates is $O({k_I}^2 \cdot D) + (k - k_I) \times O(k_I \cdot D)$, where k is the total number of landmarks.

Furthermore, we describe two ways to compute the coordinates of the secondary group of landmarks, while maintaining the same computational complexity. In the *global approach*, we compute the coordinates of each node in the secondary group relying only on the initial group as anchors. In the *incremental landmarks approach*, nodes in the secondary group are added one by one. Once a node receives its coordinate values, it becomes an anchor for all remaining nodes. To compute its coordinates, any remaining node in the secondary group can choose any k_I nodes from all embedded nodes to be its landmarks.

Landmark Selection. Finally, we consider the problem of choosing landmark nodes to produce the most accurate graph to geometric coordinate mapping. Prior work by Potamias et. al considered the problem of choosing landmarks, and concluded experimentally that choosing nodes with high centrality performed significantly better than random choice [141]. Given the complexity of computing node centrality, we consider two groups of alternative landmark selection strategies as possible approximations of centrality-based selection: Random and High-degree.

- *Random.* This is the basic landmark selection strategy. Landmarks are chosen uniformly at random from all nodes in the graph.
- *High-degree.* Prior measurements on social networks [125, 179] show that social graphs exhibit a power-law-like degree distribution. Intuitively, high degree nodes reside at the core of social graphs, effectively approximating central nodes. This strategy chooses nodes with the highest degree.
- *Landmark separation.* Closely positioned landmarks are less effective at "covering" the graph as anchors. Therefore, we add variants to the two basic strategies, where we select the landmarks one by one, ignore any potential landmarks that are too close in the graph to existing landmarks, and continue selecting landmarks until the desired number has been met.

Summary. The framework of graph coordinate systems works as a landmark-based scheme, where an initial core of 16 landmarks is first fixed in the space using Simplex Downhill optimization. A secondary group of landmarks position themselves based on the original landmarks. Finally, all remaining graph nodes

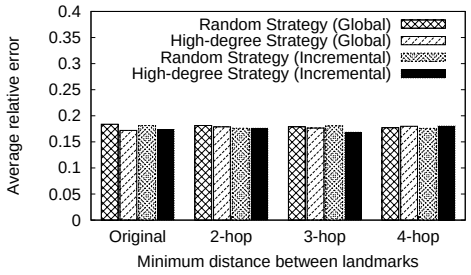


Figure 2.2: ARE of nodes' distances with different combination of landmark selection and computation strategies in India graph

calibrate their positions based on node distances obtained from computing BFS from all landmarks.

2.3.3 Evaluating Landmark Selection Strategies

We now empirically evaluate the effectiveness of the landmark selection strategies. Based on framework, we implement a prototype graph coordinate system, named *Orion* [186], which embeds node shortest path length into a Euclidean space. In the evaluations, we select 1000 random nodes and compute the pairwise distances between them with different landmark computation methods. The metric used here is the average relative error (ARE). All the experiment is repeated for 5 times and on the four largest Facebook graphs, *i.e.* Norway, Egypt, Los Angeles and India, in Table 2.1.

As the results on the four graphs are consistent, we only show India in Figure 2.2 for brevity. Figure 2.2 plots AREs for a variety of landmark selection strategies using the India graph. It shows that the graph coordinate system using Euclidean space provides low relative errors compared to actual path lengths for different landmark selection strategies. Among the considered strategies, high-

degree strategies can produce slightly lower errors. Furthermore, the impact of landmark separation on the accuracy of shortest path length estimation is fairly small. Overall, the two landmark strategies have no significant impact on the accuracy. Thus, in the later sections of this chapter, we use the Random method to select landmarks.

2.4 Embedding Shortest Path Distances

In this section, we apply graph coordinate systems to embed shortest path distances. Although the study in Section 2.3 shows that the system can accurately approximate shortest path distances using a Euclidean space, there are two key questions to be addressed when implementing a practical graph coordinate system in practice. First, among different geometric spaces, such as a Euclidean space, a spherical space and a hyperbolic space, can we find a better space in terms of the accuracy in embedding shortest path? Second, since the centralized embedding process is computationally expensive in large graphs, can we improve the embedding process to scale with large graphs?

In this section, we address the above two key questions and implement a practical graph coordinate system for shortest path distances called *Rigel*. First, we study the embedding accuracy using the three popular geometric spaces, and determine to use the most accurate space in capturing shortest path distances, *i.e.* a hyperbolic space. Second, we naturally parallelize the embedding process across servers. The embedding time can be significantly accelerated. In addition, in Section 2.4.4, we propose an approach to approximate shortest path for any node pair using graph coordinates. Comparing with the proposed algorithms,

our proposed method produces the accuracy similar to the most accurate scheme, while resolving queries up to 18 times faster.

2.4.1 Choice of Geometric Spaces

Based on the framework of graph coordinate systems, we now study the impact of geometric spaces on the estimation accuracy. First, we introduce three popular geometric spaces: a Euclidean space, a spherical space and a hyperbolic space, and empirically compute the distortion metrics [110] using different spaces.

Discussion on Geometric Spaces. *A Euclidean space* is the most widely used coordinate space. Each node in a D-dimension Euclidean space is represented by a D-dimension coordinate, *i.e.* (x_1, x_2, ..., x_D). The distance between any two nodes, A and B, is calculated by Equation 2.1. The dimension of a Euclidean space may impact the estimation accuracy of the graph coordinate system.

$$d_{AB} = \sqrt{\Sigma_{i=1}^{D}(x_i^A - x_i^B)^2} \tag{2.1}$$

A spherical space is a 3-dimension space, which is the nature representation of a sphere, such as the Earth. One representation of a node's spherical coordinate is a tuple (r, ϕ, λ), where r is the radius of the sphere, ϕ is the latitude and λ is the longitude. The distance between two nodes, A and B, in a spherical space is the shortest distance between the two nodes on the surface of the sphere, which is computed using Equation 2.2. The radius of a spherical space is an important parameter in defining a spherical space.

$$d_{AB} = r \arccos\left(\sin\phi^A \sin\phi^B + \cos\phi^A \cos\phi^B \cos\left(\lambda^A - \lambda^B\right)\right) \tag{2.2}$$

A Hyperbolic space can be thought of a space with a tightly connected core, where all paths between nodes pass through. There are five known "Hyperbolic models" that have been proposed for different purposes and graph structures, including the Half-plane, the Poincaré disk model, the Jemisphere model, the Klein model and the Hyperboloid model [153]. Each model is a different method of assigning coordinates and computing distances over the same hyperbolic structure. Since choosing a model fundamentally changes how graphs can be embedded, it is currently unknown how the choice of models affects embedding distortion.

In our design, we chose the *Hyperboloid* model for two practical reasons. First, computing distances between two points in this model is computationally much simpler than alternative models. Second, the computational complexity of calculating distances is independent of the space curvature. This gives us additional flexibility in tuning the structure of the hyperbolic space for improved embedding accuracy.

For a Hyperboloid model with curvature c, the distance between two n-dimension points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ is defined as follows:

$$\delta(x, y) = arccosh\left(\sqrt{(1 + \sum_{i=1}^{n} x_i^2)(1 + \sum_{i=1}^{n} y_i^2)} - \sum_{i=1}^{n} x_i y_i\right) \cdot |c| \qquad (2.3)$$

We empirically evaluate the accuracy of the above three graph coordinate systems using the three popular geometric spaces. Here, we use a 10-dimension Euclidean space, the best tradeoff between accuracy and efficiency [186]. In the Hyperbolic graph coordinate system, we use the curvature parameter $c = -1$ and the dimension of the space is also 10, which gives us the best accuracy in the later

Metrics	Euclidean	Hyperbolic	Spherical	Ideal Value
ARE	0.16	**0.10**	0.36	0
AAE	0.78	**0.50**	1.83	
AER	0.97	**1.00**	0.91	1
ACR	1.07	**1.02**	1.72	
ASPD	1.19	**1.11**	1.96	
SD	58.46	**30.63**	134173.04	

Table 2.2: Evaluating different embedding spaces via several metrics on the Facebook LA graph.

study. For a fair comparison, we vary the radius of the spherical space from 5 to 50, and display the best results.

We run the experiment on LA graph in Table 2.1 and use different distortion metrics [110], including average relative error (ARE), average absolute error (AAE), average expansion ratio (AER), average contraction ratio (ACR), average symmetric pair distortion (ASPD), and space distortion (SD). The results are shown in Table 2.2. Compared among the three spaces, we find that the hyperbolic space is significantly more accurate than Euclidean and spherical space in all metrics. This result is consistent previous study in network distance embedding [153, 40, 138]. An intuitive explanation is that both social graphs and the Internet topology feature highly connected graph cores, which fit the hyperbolic model well. Therefore, for the best estimation accuracy, we use the Hyperboloid model in the graph coordinate system to embed shortest path.

Optimizing Local Paths. It has been shown in Internet embedding systems [113] that the largest errors are introduced when estimating node distances for nearby nodes, *i.e.* nodes separated only by 1 or 2 hops. In addition, accuracy in resolving "local" graph queries is critical to many graph operations. In the context of graphs, this is an easy limitation to overcome, since 1-hop neighbors

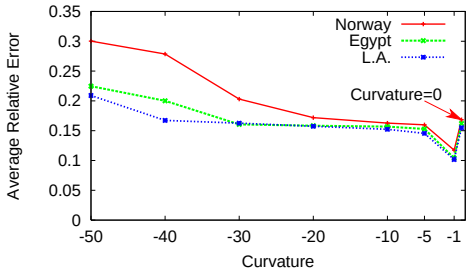


Figure 2.3: Impact of hyperbolic curvature on accuracy.

are easily accessible via graph representations, *e.g.* edge lists or adjacency matrices. The hyperbolic system uses local neighbor information to augment the node knowledge about its close-by topology. Before answering a query for a pair of nodes, it first checks their adjacency lists to detect if they are direct neighbors or 2 hop neighbors (share a node in their adjacency list), which guarantees 0 distortion in estimating 1- or 2-hop shortest path length. This optimized system is called *Rigel*.

Embedding Accuracy on Real Graphs. We now investigate how two important parameters in Rigel, *i.e.* curvature of the space c and number of dimensions of the space n, impact the embedding accuracy. Also we evaluate the efficiency of the system in terms of per-query time. We report experimental results using three Facebook datasets presented in Table 2.1, *i.e.* Egypt graph, LA graph and Norway graph.

Impact of Curvature. The curvature c of a hyperbolic space is an important parameter that determines the structure of the space. We build different Hyperbolic spaces using curvature values that range from -50 to 0, and investigate the effect on the accuracy of the distance estimation. When the curvature is 0, the

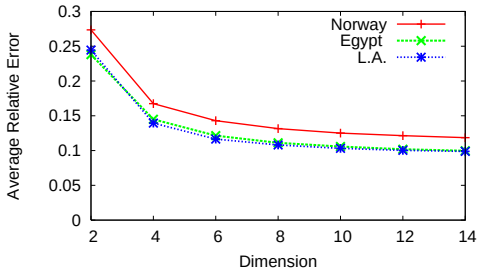


Figure 2.4: Impact of dimensionality on embedding accuracy.

hyperbolic space is equivalent to a Euclidean space. We include this value as the rightmost point in our plot. From our results in Figure 2.3, we see that the average error decreases significantly as the curvature approaches -1. We performed further fine grain tests with curvature values around -1, and find that the accuracy of our system reaches a plateau near -1. Results at curvature of -1 are 30% more accurate than results from a Euclidean system, shown in the plot as curvature of 0. Thus we use the curvature value at -1 in the rest of this chapter.

Impact of Dimensionality. The number of dimensions of a geometric space plays an important role in determining the accuracy level in the estimate of distances between nodes. Therefore, we vary the number of dimensions from 2 to 14 and evaluate the resulting accuracy. Figure 2.4 shows that increasing dimensions reduces the error from more than 0.2 to about 0.1, with most of the significant improvement occurring between 2 and 6 dimensions. Since the number of dimensions is a linear factor in the computational complexity of the Simplex method used in our embedding, we need to balance prediction accuracy against computational complexity. We find a sweet spot close to 10 dimensions, where the accuracy has essentially reached a plateau. Thus we also use 10-dimension for our hyperbolic system.

Graphs	Rigel-S	Rigel	BFS
Egypt	0.33μs	6.8μs	750,000μs
L.A.	0.33μs	8.5μs	1,027,000μs
Norway	0.33μs	17.8μs	1,440,000μs

Table 2.3: Response time for Rigel-S, Rigel and BFS.

Summary. The accuracy of the hyperbolic graph coordinate system is impacted by the choice of the space curvature c and the space dimension n. The results measured from real graphs shows that as the curvature increases, the ARE of the system decreases. While with the dimension more than 10, there is no significant improvement on accuracy. Therefore, in the remaining of this chapter, we use a 10-dimensional hyperbolic space with curvature of -1 in Rigel.

Per-Query Latency. Table 2.3 shows the average per-query response time to compute the distance of two random nodes using Rigel, and BFS. We also list the query time of Rigel without the local path optimization labeled as "Rigel-S". Since memory access in Rigel's the local path optimization adds several microseconds to each query, the per-query time of Rigel is slightly longer than Rigel-S (see the second column). But overall, Rigel per-query time is still 5 orders of magnitude faster than BFS.

2.4.2 Parallelizing Embedding Process

Since the complexity of Rigel embedding scales linearly with graph size, this processing overhead presents a significant performance bottleneck for large graphs with millions of nodes, and prevents practical applications of Rigel on large social graphs. Here, we describe a mechanism to address this challenge by parallelizing

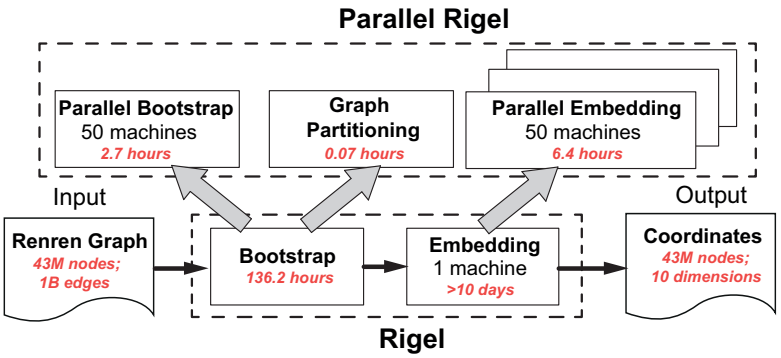


Figure 2.5: A high-level view of how embedding is parallelized and its net impact on embedding latency for Renren, our largest graph.

Rigel's embedding process across multiple servers, named as "Parallel Rigel". We then evaluate its impact using four large social graphs.

Parallel Rigel. We integrate the parallel mechanisms with the original Rigel design, called *Parallel Rigel*. Figure 2.5 demonstrates the Parallel Rigel system on top of and contrasts it to the original Rigel design. It consists of three components: *parallel bootstrapping*, *graph partitioning* and *parallel embedding*. The parallel bootstrapping module distributes BFS tree computation related to each landmark across servers, one or more landmarks per server. The graph partitioning module provides a balanced distribution of nodes across servers. The cost of this operation is negligible since simple partitioning schemes are sufficient. Finally, the parallel embedding module embeds all graph nodes in parallel across the servers, allowing Parallel Rigel to achieve significant speedup.

Computational Efficiency Evaluation. We have implemented a fully functional prototype of parallel Rigel, and used it to four of the largest social graphs available today, Flickr, Orkut, Livejournal and Renren in Table 2.1, to examine

Graphs	Bootstrap (hours)		Graph Partitioning (hours)	Embedding (hours)		Response	
	Rigel	P-Rigel	P-Rigel	Rigel	P-Rigel	BFS	Rigel
Flickr	1.4	0.028	0.003	9.7	0.24	24,500,000μs	12.9μs
Orkut	7.5	0.15	0.005	19.4	0.42	56,200,000μs	36.6μs
Livejournal	4.8	0.096	0.008	32.2	0.66	65,200,000μs	8.4μs
Renren	136.2	2.7	0.07	250	6.4	1,598,000,000μs	28.9μs

Table 2.4: Comparing the time complexity of Rigel and Parallel Rigel (P-Rigel) using a cluster of 50 servers.

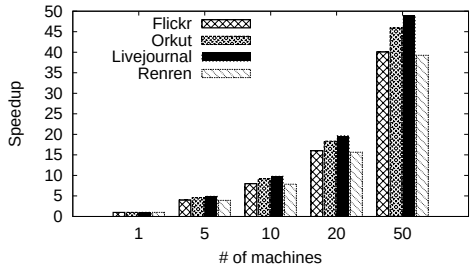


Figure 2.6: Average speedup achieved by Parallel Rigel on different cluster configurations.

the efficiency of Parallel Rigel. All the experiment is tested on a cluster of 50 servers (Dell Xeon, 2GB).

We evaluate the efficiency of Parallel Rigel by comparing its computation time to that of original Rigel. By utilizing a cluster of servers, Parallel Rigel distributes the tasks of landmark bootstrapping and graph embedding over multiple parallel servers. While Parallel Rigel does require an extra step of graph partitioning by distributing nodes among machines, it only leads to a minor increase in time complexity, less than 0.1% of the original bootstrapping time. Table 2.4 shows the comparison when Parallel Rigel runs on a cluster of 50 servers. We see that Parallel Rigel achieves close to linear speedup, even slightly better due to reduced virtual memory paging on each server.

Metric	Method	Egypt	L. A.	Norway	Flickr	Orkut	Livejournal	Renren
Radius	Ground Truth	**9**	**11**	**8**	**13**	**6**	**13**	**12**
	Rigel	8.7	11.0	7.5	12.7	6.4	12.2	12.0
	Orion	9.2	10.7	7.8	12.6	6.3	12.0	12.1
Diameter	Ground Truth	**14**	**18**	**12**	**19**	**8**	**17**	**15**
	Rigel	14.8	17.9	11.7	18.6	10.2	17.7	14.9
	Orion	14.4	17.8	12.2	17.3	10.0	16.8	14.9
Average Path Length	Ground Truth	**5.0**	**5.2**	**4.2**	**5.1**	**4.1**	**5.4**	**5.0**
	Rigel	4.9	5.1	4.2	5.0	4.3	5.5	4.9
	Orion	4.7	5.0	4.1	4.3	3.9	4.8	4.6

Table 2.5: Comparing separation metric results, as computed by Rigel, Orion, and BFS (ground truth).

To examine the impact of the cluster size, we compare the speedup of Parallel Rigel by using 5, 10, 20 and 50 servers, where speedup is the decrease in embedding time. Figure 2.6 shows that run time decreases almost linearly with cluster size.

2.4.3 Evaluating System Accuracy in Applications

In this section, we implement three path-length based applications, *i.e.* separation metric computation, graph centrality computation and distance-ranked social search, and evaluate the performance of using the coordinates generated by Rigel. In each case, we compare the accuracy of using Rigel against that of Orion, the prototype system proposed in Section 2.3.

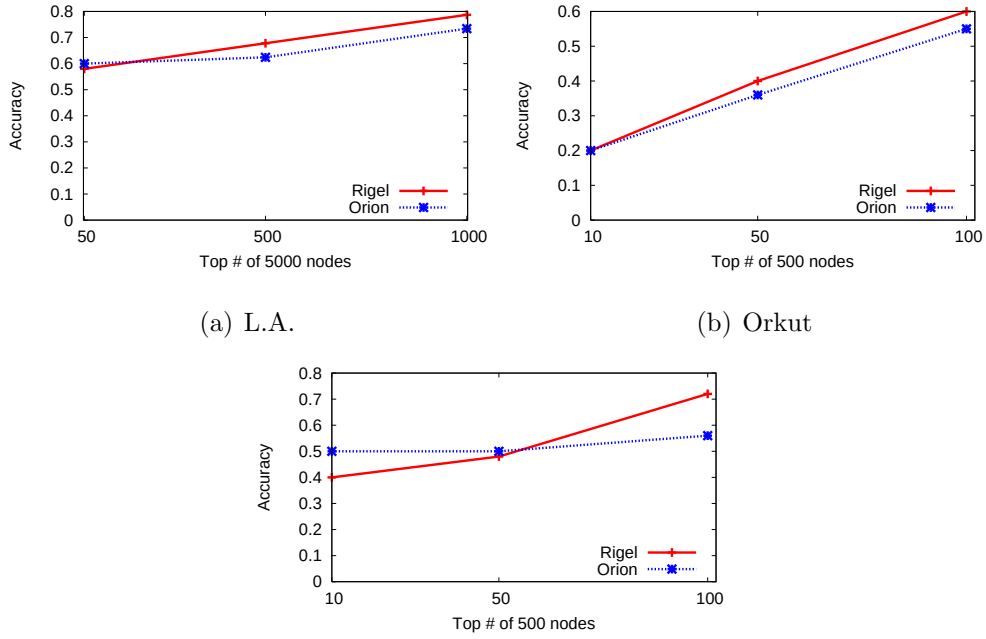
Computing Separation Metrics Social network graphs are known for displaying the "Small World" behavior. Graph separation metrics such as diameter, radius and average path length, have been widely used to examine and quantify the Small World behavior. But since each of these metrics relies on large numbers of node distance computations, computing them for large graphs can become extremely costly or even intractable.

Using Rigel, we build an application to compute the graph separation metrics listed above, and examine their accuracy by comparing their results to ground truth. Since computing shortest path length between all node pairs takes several days even for our smallest graph (Facebook Egypt), we take a random sampling approach to compute the ground truth. We randomly sample 5000 nodes from the three Facebook graphs, 500 nodes from Flickr, Livejournal and Orkut, and 100 nodes from Renren, and use shortest path lengths between these pairs to derive the separation metrics.

We report the results in Table 2.5 for Radius, Diameter and Average Path Length on seven different graphs, for Rigel, Orion and Ground Truth. In general, Rigel consistently provides more accurate results compared to Orion. More importantly, Rigel provides results across all three metrics that are extremely close to ground truth values.

Computing Graph Centrality. Graph centrality is an extremely useful metric for social applications such as influence maximization [33] and social search. For example, application developers can use node centrality values to identify the most influential nodes for propagating information in an online social network. Formally, the most "central" node is defined as the node that has the lowest average node distance to all other nodes in the network.

Using Rigel, we implement a simple application to compute node centrality directly from node distance values, where a node with a small average path length has a high centrality score. As before, we examine the accuracy of our Rigel-enabled application by computing the centrality of $x = 5000$ randomly chosen nodes on the three Facebook graphs, *i.e.* Egypt, LA and Norway, $x = 500$ ran-



(a) L.A.

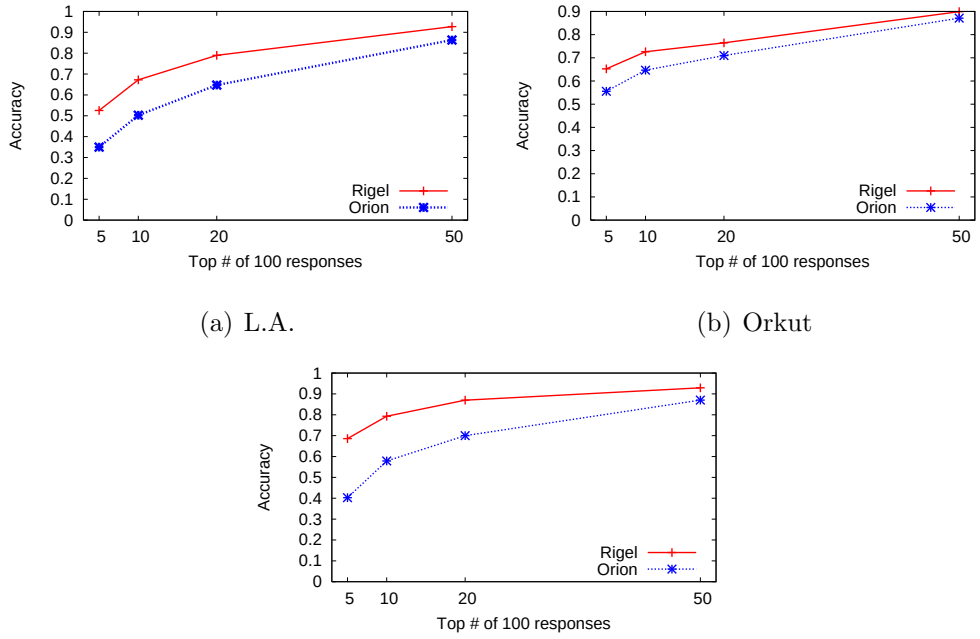
(b) Orkut

(c) Livejournal

Figure 2.7: Average accuracy of queries for the top k high centrality nodes. Rigel consistently outperforms Orion.

domly chosen nodes each for Flickr, Livejournal Orkut, and $x = 100$ nodes for Renren. For each graph, we sort these x nodes by centrality, and select the top k nodes. We compute the "accuracy" of Rigel's centrality ordering by counting the number of overlapping nodes (m) in Rigel's top k nodes and actual top k centrality nodes as computed by BFS on the original graph. We study the accuracy of our Rigel-based system as the ratio of m to k.

We perform our experiments on the seven of our social graphs, and find the results to be highly consistent. For the rest of this section, we will only report results for three of them: Facebook LA, Orkut and Livejournal. Figure 2.7 shows the centrality accuracy results for different values of k. As expected, the accu-



(a) L.A.

(b) Orkut

(c) Livejournal

Figure 2.8: Average accuracy of social search queries that return top k ranked nodes

racy of both Rigel and Orion increases with larger k values. In general, Rigel consistently outperforms Orion for different graphs and different values of k.

Distance-Ranked Social Search. Social networks such as Facebook and LinkedIn can best serve their users by ranking search results by the proximity of each result to the user in the social graph [123]. This is because users are likely to care about its social proximity to the origin of the search result as much as the quality of the result itself, *i.e.* a user would pay more interest to results from her close friend rather than those from an unrelated stranger.

Despite its usefulness, using social distance in search results is highly costly due to the number of node distance computations necessary for each social search

query. Instead, we can leverage Rigel's constant time node-distance functionality to build powerful distance-based social search applications.

To verify the impact of Rigel on distance-ranked social search, we perform the following experiment. For each node initiating a query, we select 100 random nodes in the graph to respond to the query. We sort the responses by their distances to the query node, computed via Rigel and Orion, and return the top k nodes to the user. We then compute the same top k nodes using BFS for distance computation, and examine the percent of overlapping nodes between the result sets as a measure of accuracy. We repeat this experiment 5000 times on smaller graphs, *e.g.* Facebook Egypt, LA and Norway, and 100 times on our largest graph, *i.e.* Renren. We vary k from 5 to 50, and show the results of L.A, Orkut and Livejournal in Figure 2.8. It shows that Rigel's hyperbolic coordinates consistently and significantly outperform Orion's Euclidean coordinates. On Livejournal, for example, when we rank the top 5% search results, average accuracy of Rigel is 70% while Orion only achieves 40%.

2.4.4 Finding Shortest Paths Using Graph Coordinates

A number of critical graph-based applications require not only the length of the shortest paths, but also the actual shortest path between two nodes. For example, on the Overstock social auction system, users can search how they connect to the seller of a given object, and choose to buy from friends of friends instead of complete strangers [159].

In this section, we describe a novel extension to Graph Coordinate Systems that produces accurate approximations of shortest paths by using node distance

queries as a tool. We first describe how this extension to Rigel computes short paths between any two nodes. Next, we describe the Sketch algorithm [42], an efficient algorithm for shortest path estimation, and its followup algorithms including SketchCE, SketchCESC, and TreeSketch [67]. Finally, we compare Rigel's shortest path algorithm against these algorithms on a variety of social graphs in both accuracy and per-query runtime. We show that while Rigel requires similar preprocessing times to these algorithms, Rigel's shortest paths return query results 3-18 times faster, while matching the best of these algorithms in accuracy.

An Algorithm to Find Shortest Paths Using Graph Coordinates. We now describe a heuristic that uses our coordinate system to find a good approximation of the shortest path connecting any two nodes. Our algorithm, which we call *Rigel Paths*, uses techniques reminiscent of the routing algorithm in [138].

Given two nodes A and B, we start by computing the distance between them $d(A, B)$. If the distance is 1 or 2 hops, we can use simple lookup on their adjacency lists to determine the shortest path between them. If the predict distance between the nodes is greater than 2 hops, then we begin an iterative process where we attempt to explore potential paths between the nodes using the coordinate space as a directional guide.

Starting from A, we use Rigel to estimate the distance of each of its neighbors N_i^A to B. The expected distance for a neighbor on the shortest path should be $d(A, B) - 1$. If any neighbor's estimated distance is within a δ factor of that prediction, it is considered a candidate to explore. For each of A's neighbors that qualify as a candidate node, we repeat the process to obtain candidates for hop 2. This process iterates until one of the candidate nodes is a direct neighbor of B.

At each iteration of the algorithm, *i.e.* for the n^{th} hop, we keep a maximum number of candidates C_{max} to explore. Choosing this number manages the trade-off between exploring too many paths (and extending processing latency) and exploring too few paths (and finding a dead end or inefficient paths). In practice we choose C_{max} to be 30, and δ to be 0.3.

Sketch-based Algorithms for Shortest Path. We first describe the state-of-the-art algorithms for locating shortest paths. There are four algorithms all based on variants of the Sketch algorithm [42, 67].

Sketch [42]. Sketch is a landmark-based solution where each node computes its shortest paths to the landmarks and then uses common landmarks between itself and another node in the graph to estimate their shortest paths. This method selects $r = \lfloor \log N \rfloor$ sets of landmark nodes, where N is the number of the graph nodes. For each node, Sketch computes its shortest paths to k (k=2) different landmarks in each set. Those shortest paths are precomputed by using the results of BFS trees rooted in each landmark. Therefore, for an undirected graph, each node is associated with $k \cdot r$ shortest paths.

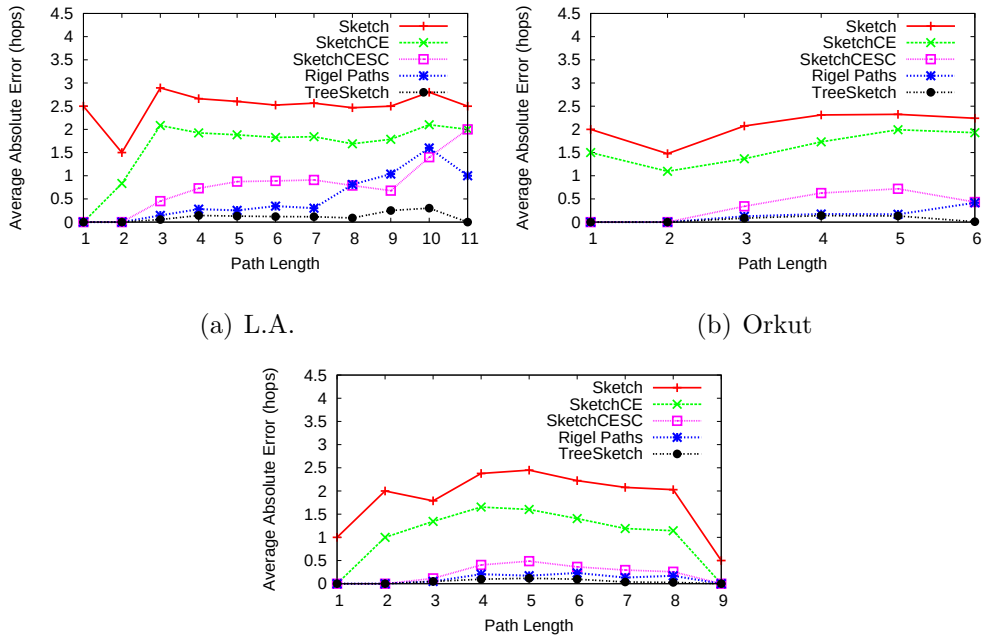
Cycle Elimination, Short Cutting and TreeSketch [67]. These three algorithms are variants of the basic Sketch approach for finding shortest paths [67]. *First,* Cycle Elimination, called SketchCE, improves Sketch by simply removing cycles in the estimated paths computed by Sketch. *Second,* Short Cutting improves Sketch by searching for bridging edges between two nodes x and y, where x is on the path between the source and the landmark and y is on the path from the landmark to the destination. If such an edge is found, this edge replaces the sub-path through

the landmark. This approach is called as SketchCESC. It locates shorter paths, but dramatically increases computational time.

Finally, TreeSketch is a tree-based approach. At query time, TreeSketch builds two trees separately rooted in the source and the destination using precomputed paths to landmarks. Given the two trees, the path search starts from both root nodes, and iteratively explores more nodes from both trees. BFS computation starts from roots of both trees. For each visited node u in a tree, its neighbors are compared with any visited node v in the other tree. Once a common node is found, the shortest path between source and destination is constructed using the sub-path from source to node u, the edge (u, v), and the sub-path from v to the destination. While producing very accurate paths, TreeSketch is computationally slow due to the tree construction and extensive search process.

Comparing Shortest Path Algorithms. We compare our *Rigel Paths* to Sketch, SketchCE, SketchCESC and TreeSketch in accuracy and query latency.

Experimental Settings. To compare Rigel Paths against prior work, we obtained the source code for the sketch-base algorithms from the authors [67]. The code runs on RDF-3X [129], a specialized database system optimized for efficient storage and computation of large graphs. We run both Rigel Paths and sketch-base algorithms on seven graphs in Table 2.1, including Egypt, LA, Norway, Flickr, Orkut, Livejournal and Renren. All experiments were performed on Dell quad-core Xeon servers with 24GB of RAM, except for Renren experiments, which were performed on similarly configured Dell servers with 32GB of RAM.



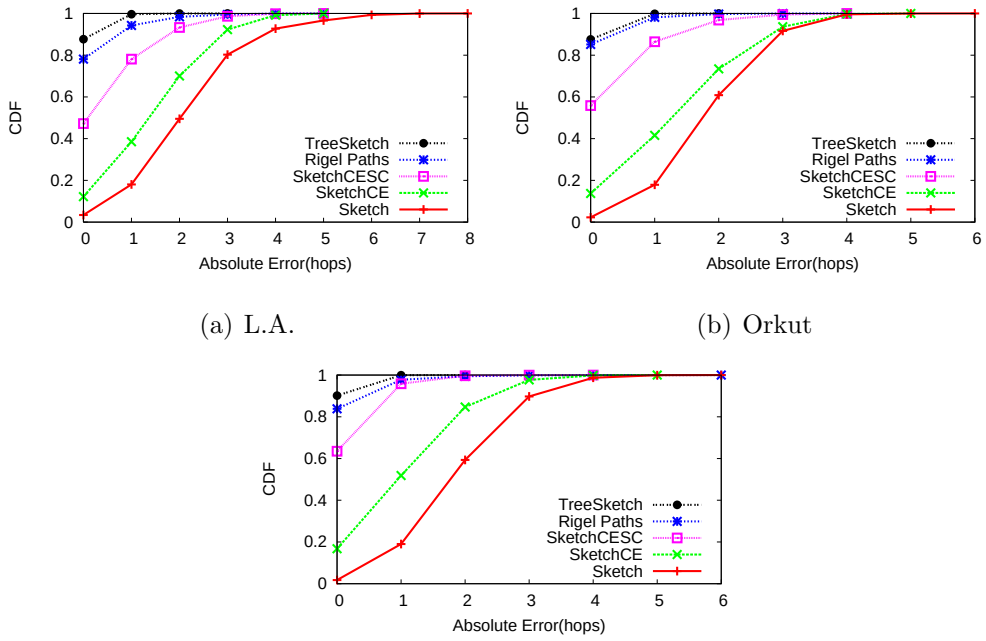
(a) L.A. (b) Orkut

(c) Livejournal

Figure 2.9: Absolute error (in hops) of shortest paths returned by Rigel Paths, Sketch, SketchCE, SketchCESC and TreeSketch.

Accuracy. For the above seven graphs, we randomly sample 5000 node pairs, and compare the shortest path results of Rigel Paths, Sketch, SketchCE, SKetchCESC, and TreeSketch algorithms against the actual shortest paths computed via BFS.

Figure 2.9 shows the average absolute error of the five different algorithms broken down by length of the actual shortest path. Here we define the absolute error as the additional number of hops in the estimated path compared to the real path. As before, we only show the Facebook Los Angeles, Orkut and Livejournal graphs for brevity, because their results are representative of results on other graphs. The results show consistent trends across the graphs. The Sketch and SketchCE algorithms are highly inaccurate, and generally produce shortest paths



(a) L.A.

(b) Orkut

(c) Livejournal

Figure 2.10: CDF of the absolute error in path finding among Rigel Paths, Sketch, SketchCE, SketchCESC and TreeSketch.

that are roughly 2 hops longer than the real path. TreeSketch and Rigel Paths are the most accurate algorithms and often indistinguishable from each other.

We show the CDF of absolute errors of the different algorithms in Figure 2.10. This shows a clear picture of the distribution of errors. Rigel paths and TreeSketch are by far the most accurate algorithms. Both produce exact shortest paths for a large majority of node pairs. Both are significantly better than SketchCESC. SketchCE and Sketch are fairly inaccurate, and provide paths with multiple hop errors for the overwhelming majority of node pairs. While Rigel Paths provides accuracy that matches or beats all of the Sketch based algorithms, we will show later that it is significantly faster than both SketchCESC and TreeSketch (ranging from a factor of 3 to a factor of 18 depending on the specific graph).

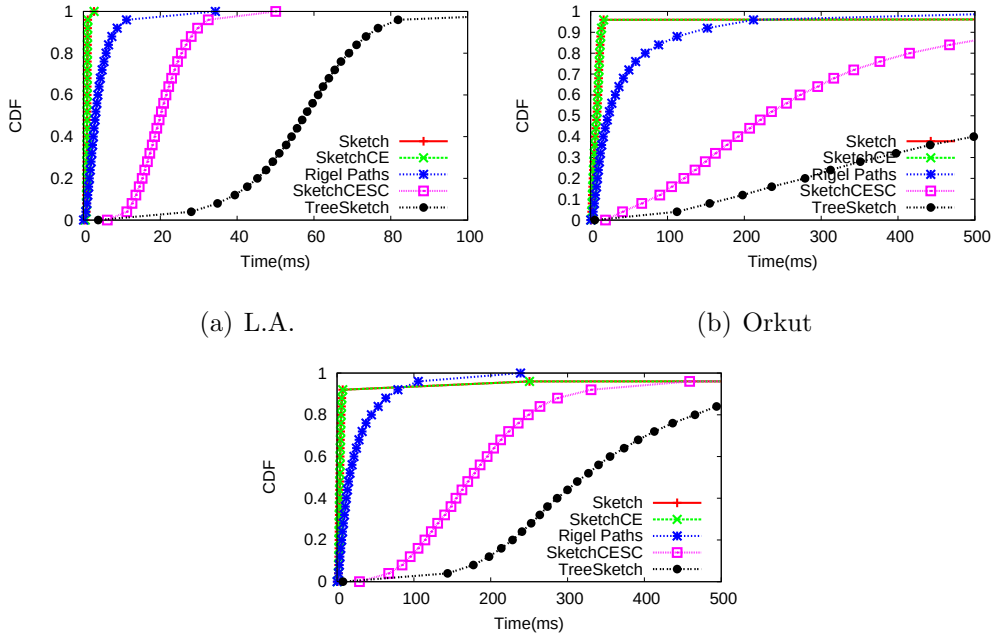
Graphs	Preprocessing (Hours)		Per-Query Response Time (μs)					
	Rigel	Sketch	Rigel	Sketch	SketchCE	Rigel Paths	SketchCESC	TreeSketch
Egypt	1.3	0.43	6.8	1781	1792	3667	38044	62407
L.A.	1.5	0.54	8.4	936	946	4008	20597	56828
Norway	1.4	0.67	17.8	1492	1501	4621	21472	59635
Flickr	9.7	3.3	12.9	17157	17178	41279	732332	630890
Orkut	19.4	13.1	36.6	21043	21054	49470	273586	730284
Livejournal	32.2	14.2	8.4	75101	75114	28355	253976	348464
Renren	250	348	28.9	124327	124334	181814	546925	2594756

Table 2.6: Comparing the preprocessing times and per-query response times of Rigel Paths, Sketch and variants SketchCE, SketchCESC and TreeSketch. Preprocessing/embedding time for Rigel (and Rigel Paths) is for single server (non-parallel version).

Computational Costs. We now compare Rigel Paths and the four Sketch algorithms on computational time complexity. We break down our analysis into two components. First, we measure each algorithm's *preprocessing time*. For Rigel Paths (and Rigel), this represents the time required to embed the graph into the coordinate space. For all Sketch algorithms, this is the time to compute shortest paths (using BFS) to all of their landmark nodes [67]. Our second component measures the computational latency required to resolve each query. All experiments are run on a single server.

In Table 2.6, we see that Rigel takes roughly 2–3 times longer preprocessing time. Note, however, that these measurements were run on only a single server. As shown in Figure 2.6, we can distribute Rigel's preprocessing phase across multiple machines with close to linear speedup. Thus, we can reduce Rigel preprocessing by spreading the load over 2 or 3 machines.

Again, we choose 5000 random node pairs in each graph, and compare the average query response time for each algorithm in Table 2.6. Recall that Sketch and SketchCE produce paths that are highly inaccurate, *i.e.* introduce an average of 2-3 additional hops in each path. Of the two best algorithms, Rigel Paths



(a) L.A.

(b) Orkut

(c) Livejournal

Figure 2.11: CDF of computing time in path finding among Rigel Paths, Sketch, SketchCE, SketchCESC and TreeSketch.

and TreeSketch, Rigel paths returns results in a fraction of the time required by TreeSketch and SketchCESC. The latency reduction ranges from ∼3 (against SketchCESC on Renren) to a factor of 18 (against SketchCESC on Flickr). We show a CDF of these results in Figure 2.11. Rigel Paths is clearly much faster than both TreeSketch and SketchCESC.

Finally, we also include the node-distance computation time from Rigel as a point of reference. Clearly, finding actual shortest paths is orders of magnitude more expensive than simply computing node distance. Luckily, the large majority of graph analysis tasks only require node-distance computation, and only user-interactive queries require the full shortest path between node pairs.

2.5 Embedding Random Walk Distances

Compared to shortest path length, random-walk distances, such as hitting time, commute time, and PPR, are more effective metrics to measure node similarity in a graph. This comes from the fact that in addition to the node distance, random-walk distances capture the number of paths. Therefore, random-walk distances are widely used in quantifying user similarity in social networks [107] or measuring web proximity in search engine [95].

However, because of the inherent randomness, computing random-walk distances is a computational costly process. In today's graphs with millions of nodes, computing hitting time between a single pair of nodes takes minutes or even an hour. This prevents random-walk distances from being used in practice.

As shown in Section 2.3, graph coordinate systems are an alternative approach to capture and estimate shortest path length in microseconds. In this section, we explore whether graph coordinate systems can be used to embed random-walk distances with high accuracy. First, we introduce three popular random-walk distance metrics, including hitting time, commute time and PPR. We also identify two key challenges in embedding them on the basic graph coordinate systems, *i.e.* the asymmetry of random-walk distances and high cost of precomputation. To solve the two challenges, we design a new graph coordinate space that explicitly accounts for asymmetry in random walks in Section 2.5.2, and propose simple techniques that generate ground truth samples with low computation cost in Section 3.4.2. Finally, by experimenting on various graphs from different networks, it is shown that with low computational cost, the new space not only accurately captures asymmetric distances, but also significantly improves the accuracy of

symmetric distance embedding. In addition, we use two application level tests to demonstrate that our embedding causes very small deviations in the results produced by applications.

2.5.1 Random-Walk Distances and Challenges

In this section, we first define in detail random-walk distances in undirected unweighted graphs, including hitting time, commute time and personalized PageRank. We then identify the two challenges that arise when this approach is applied to random-walk distances.

Random-walk based Distances. In undirected unweighted graphs, a random walk is a sequence of random steps. Consider an undirected unweighted graph G, with vertices V and edges E. Starting from node v_0, a random walk in G chooses its next destination: if we are at node v_k at the k^{th} step, we randomly select a neighbor v_{k+1} of v_k with probability $1/d(v_k)$ as the destination of the $(k+1)^{th}$ step, where $d(v_k)$ is the degree of node v_k. Thus, the sequence of random nodes $v_k(k = 0, 1, 2, \cdots)$ is a random walk from node v_0 in Graph G.

There are a number of distance measures based on random walks. Our work focuses on the three most popular random-walk distances, *Hitting Time*, *Commute Time* and *Personalized PageRank (PPR)*.

Hitting Time. Hitting time from node i to node j is the expected number of hops in a random walk starting from node i before it reaches node j for the first time. Since graph density and local structure around nodes i and j are different,

hitting time from node i to node j is likely different from the hitting time from node j to node i. In other words, hitting time is asymmetric.

Commute Time. Commute time between two nodes i and j is the expected number of random walk hops from node i to j and then back to node i. Thus commute time is the sum of two hitting time distances, one from i to j and one from j to i. Thus, commute time is symmetric.

Personalized PageRank. Personalized PageRank (PPR) [95, 77] from node i to node j is the likelihood that a random walk starting from node i ends at node j with the reset probability α. In a random walk with reset, the reset probability α is the probability that at each hop, a node v can choose to selects itself as its next random walk step (*i.e.* resets its walk). In each step at node v_k, the random walk selects the current node v_k as the next step with reset probability α, and uniformly selects one of its neighbors with probability $1 - \alpha$. By starting m such random walks originating at node i, we count the number of random walks ending at node j on the T^{th} step (m_j), where T is a parameter chosen to capture the number of hops before the random walk probability converges for any given destination. PPR from node i to j is the ratio m_j/m.

All three distance measures have been used extensively in different contexts. Despite the simplicity, these measures are very powerful because they are able to incorporate two fundamental properties behind the affinity of pairs of nodes in graphs - specifically the node distance, as well as the number of paths. In particular, the similarity between two nodes based on any of the three aforementioned random walk measures is likely to be higher if the two nodes are *closer* in the node

distance sense. Alternatively, given the same node distance between two nodes, at a very high level, the random-walk based similarities are likely to be higher when *multiple paths* exist between these nodes. From an application standpoint, notice that any path between two nodes is a weak signal of similarity, and with multiple paths, any such reasonable notion of similarity should be reinforced. Similarly, a short path means the two nodes are close/similar.

Ground Truth Computation. In this section, we will study the embeddability of the three random walk distances. One challenging component of the embedding process is computing ground truth values of the random-walk distances between landmarks and regular nodes. Our solution is brute force search, where we simulate multi-round random walks on each social graph and derive the mean values.

Hitting time computation. The hitting time from node i to j, $H(i, j)$, is the expected number of random walk hops from i to j. To compute $H(i, j)$, we simulate a random walk starting from i until it reaches j for the first time, repeat the process N times, and compute the average of the hop count from each walk. We choose $N = 2000$ because our experiments show that the average hop count stabilizes at this value.

Commute time computation. Once we measure the hitting time from node i to node j and the hitting time in the other direction, *i.e.* $H(i, j)$ and $H(j, i)$, we can easily derive the commute time between node i and j, $C(i, j) = H(i, j) + H(j, i)$.

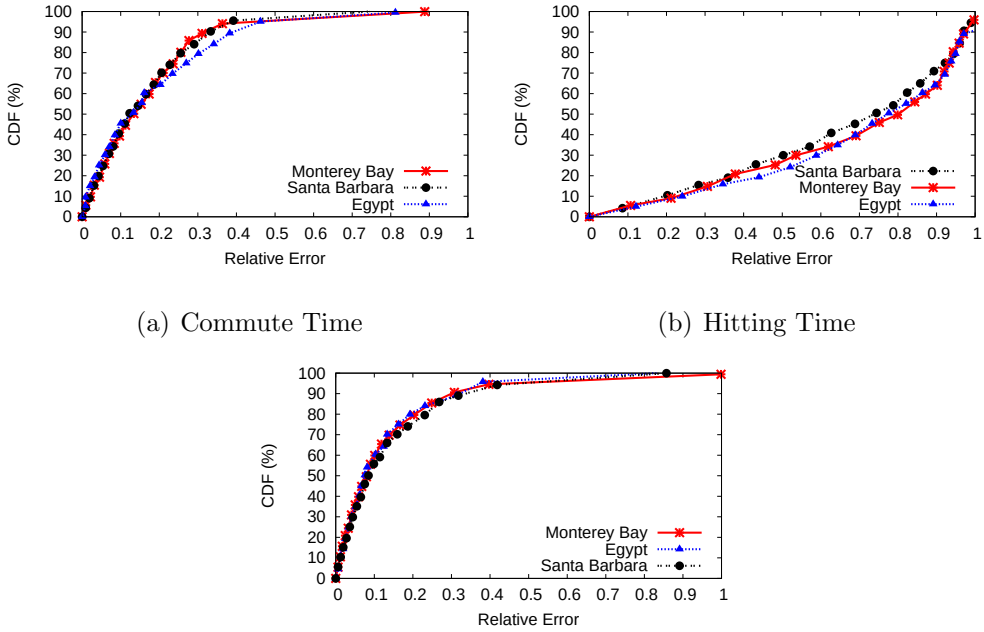
PPR computation. We initiate a random walk from node i with reset probability α, terminate the walk at the $T^t h$ step, and repeat the process for N times. We then compute m_j, the number of times that a node j is visited at

the T^th step across all N rounds. The PPR from node i to j is computed as $PPR(i,j) = m_j/N$. Our experiments use $\alpha = 0.15$, the common choice of PPR computations [77, 17], and $T = \log n/\alpha$ because prior work proved that PageRank converges in $O(\log n/\alpha)$ hops [43]. We also found that $N = 8000$ is adequate to get a stable PPR estimation.

Unlike hitting time and commute time, PPR cannot be directly embedded using graph coordinates. This is because of two reasons. First, the embedding process assumes when two nodes are close to each other in the embedded graph, their actual distance is also small. This is true for shortest path, hitting time and commute time, but not for PPR. The larger the PPR value, the more similar (and thus closer) the two nodes. Second, the value of PPR is always between 0 and 1, a range that cannot be accurately captured by graph coordinate systems. We address these two issues by embedding an alternative metric $(1 - PPR(i,j)) \cdot 10^6$ instead of $PPR(i,j)$ itself.

Challenges in Embedding Random-Walk Distances. Our goal is to test the feasibility of using geometric space embeddings to capture random-walk distances. Specifically, we look for a graph coordinate system that maps nodes in a graph into a geometric space of fixed dimensions, where distances between nodes represent estimated values of expected random-walk distances. Using a node's coordinate position in the space, we can accurately estimate the corresponding random-walk distances between any two nodes in constant time.

A naive solution is to apply the design of current graph coordinate systems, and substitute random-walk based distances for shortest path distances. However, two key properties of random-walk based distances pose real challenges and prevent



(a) Commute Time

(b) Hitting Time

(c) PPR

Figure 2.12: Relative error CDF of random-walk distances embedding using Orion in Facebook graphs

us from using this naive approach. We explore and summarize these two issues below.

Asymmetry. The first and most critical difference between random-walk distances and shortest path distances is symmetry. In undirected graphs, shortest path length is symmetric by definition. In contrast, hitting time and PPR are asymmetric [112, 77], *i.e.* distance (either hitting time or PPR) from node A to B may not be the same as the distance in the reverse direction. In addition, distances in any geometric space, *e.g.* Euclidean space or hyperbolic space, are symmetric. This leads us to believe that embedding random-walk distances on coordinate spaces will produce significant errors.

To prove our conjecture, we examine whether we can simply apply existing graph coordinate systems to embed random-walk based distances. Using 11 graphs in Table 2.1[2], we evaluate the embedding accuracy using two graph coordinate systems based on traditional geometric spaces, *i.e.* Orion using a Euclidean space and Rigel using a hyperbolic space. As the Euclidean space is more accurate when embedding high-variance metrics like hitting time and commute time, we only show the embedding accuracy using Orion.

Since the results are consistent on different graphs, we only use the results of the three Facebook graphs in Figure 2.12 to demonstrate the embedding accuracy. As expected, the relative error in embedding symmetric commute time is low in Figure 2.12(a), while the accuracy of the embedded asymmetric hitting time is significantly larger in Figure 2.12(b).

An interesting observation is that while PPR is also an asymmetric metric, its embedding error is similar to that of commute time. This is because an inherent artifact of the PPR computation. Our measurement shows that 63.6% node pairs in the three graphs have zero PPR for both directions, while for the rest, the degree of asymmetry, *i.e.* the relative difference for PPR in both directions, is less than 0.006. This means PPR in fact becomes a symmetric metric, and explains why the embedding performance is closer to that of commute time.

The experiments show that existing graph coordinate systems based on traditional geometric spaces embed commute time and PPR at a reasonable accuracy similar to that of shortest path, but produces large errors on asymmetric distances

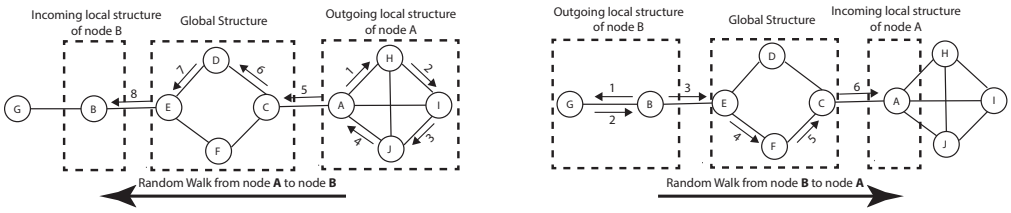
[2]The 11 graphs includes 3 different size Facebook graphs, *i.e.* MontereyBay, Santa Barbara and Egypt, 7 graphs from different non-social networks and the synthetic planar graph

like hitting time. This motivates us to search for a new space in Section 2.5.2 to properly capture both symmetric and asymmetric distances.

Cost of precomputation. Second, we note that it takes significantly more time to obtain ground truth of random-walk based distances, especially for hitting time and commute time. Depending on graph structure, arriving at a stable expected value for random walks can require thousands of independent random walks. For instance, using a commodity server with sufficient main-memory, computing expected hitting time from one landmark node to all nodes in a $250K$-node social network graph takes 60 days. In contrast, it takes only 2 hours to compute the shortest path distance (using BFS) between 100 landmarks and all nodes in the graph. Therefore, to make any embedding system practical for random-walk distances, we also need to address the issue of efficiently obtaining ground truth. We address this issue further in Section 2.5.3.

2.5.2 A Directional Height Space

Our experiments in Section 2.5.1 show that a traditional embedding system produces significant errors when estimating random walk distances, especially asymmetric distances. In this section, we present a new graph coordinate space, a directional height space, which explicitly accounts for asymmetry in random walks. The intuition behind our design is that asymmetry in random walks is caused by distinct "local" graph density around each node, and by capturing such effect on a per-node basis, one can effectively model random walks via graph coordinates.



(a) Random Walk from Node A to B (b) Random Walk from Node B to A

Figure 2.13: Example of a random walk from node A to B in (a) and a random walk from node B to A in (b).

Our discussion begins by analyzing the cause of asymmetry in hitting time based random walks, where we illustrate the significant effect of local graph density. We then present a new coordinate space combining Euclidean coordinates and heights to capture such effect on a per-node basis, followed by a description of the overall embedding process.

A Closer Look at Hitting Time. To illustrate the cause of asymmetry in hitting times, we consider a toy example using random walks between node A and B in Figure 2.13. In this example, an arrow from node i to node j represents a random walk step from i to j, and the number k on top of the arrow represents the sequential order of the current random walk, *i.e.* the k^{th} step.

Figure 2.13(a) shows an instance of random walk from node A to B. Since A's neighborhood is tightly connected, *i.e.* a clique consisting of A, H, I and J, it takes 5 hops to leave A's local structure and reach node C. The subsequent random walk takes 2 hops to reach node B's local neighborhood E, and another 1 hop to reach B. In total, the random walk takes 8 hops. Figure 2.13(b) illustrates the random walk from node B to A. Here node B only has two neighbors, and the current instance of random walk takes 3 hops to leave B's local neighborhood.

It takes another 2 hops to reach C, and another extra hop to reach A. In total this random walk from B to A only requires 6 hops, 2 hops less than that from A to B.

This example also sheds light on one potential view of why random walks are asymmetric. We can think of random walks as traversing through three abstract "regions" of the graph: first exiting an *outgoing local structure* near the source node, moving across a *backbone global structure* in the graph, and finally finding the destination node inside its an *incoming local structure*. Our intuition into the asymmetry of hitting time is that random walk distances are largely dominated by a node's incoming and outgoing local structures, while traversal across the global graph structure can be thought of as fairly symmetric. For example, both node A and B's incoming (Figure 2.13(b)) and outgoing (Figure 2.13(a)) local structures are significantly different, leading to the large difference of 2 hops between their corresponding average random walk distances. Clearly, these differences cannot be captured by traditional embeddings.

Per-Node Height Vectors. The above intuition implies that we need a graph coordinate system with three components, two asymmetric components that capture each node's local structure for outgoing and incoming random walks, and a symmetric component that captures the global structure. Coordinate systems (*i.e.* Euclidean, Hyperbolic or Spherical spaces) can easily capture the symmetric component. Our task is to identify and model the remaining two directional asymmetric components. For this, we introduce *directional height vectors*. More specifically, we use *two* distinct height vectors for each node, $h_{in}(i)$ and $h_{out}(i)$

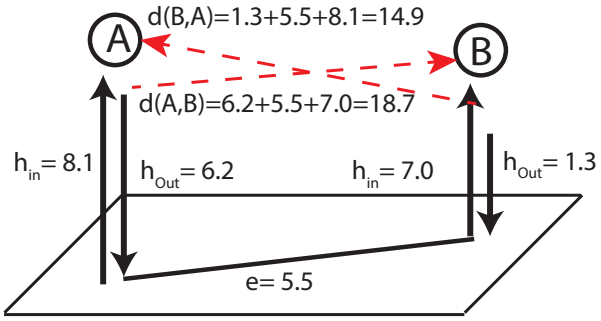


Figure 2.14: An example of two nodes in our new coordinate space composed of the 2D Euclidean space and two heights. The vertical lines represent height vectors, and the arrows mark the directionality (incoming/outgoing). The line e represents the distance in the Euclidean space, and the red dashes represent the predicted random walk distances produced by our system. Note that a node's outgoing vector is typically smaller than its incoming vector.

($h_{in}(i) \geq 0$, $h_{out}(i) \geq 0$), to represent the asymmetric outgoing and incoming local structure for a node in random walks.

Our embedding system for random walks computes predicted distances by combining two appropriate height vectors with an undirected distance captured by the baseline embedding space. As shown in Figure 2.14, a random walk from node A to B will first exit its local structure with outgoing height $h_{out}(i)$, followed by the core global structure represented by a Euclidean distance between A and B, and finally through the local structure of j with incoming height $h_{in}(j)$. The total expected random walk length, *i.e.* predicted hitting time, is the sum of the Euclidean distance and two heights, $h_{out}(i)$ and $h_{in}(j)$:

$$d(A,B) = h_{out}(A) + \sqrt{\sum_{i=1}^{n}(x_i(A) - x_i(B))^2} + h_{in}(B) \tag{2.4}$$

where the vector $\{x_i\}_{i=1}^{n}$ represents the n-dimension Euclidean coordinates of node i.

The example in Figure 2.14 shows a basic 2-dimensional Euclidean plane coordinate space. The heights of node A are $h_{in} = 8.1$ and $h_{out} = 6.2$, and those for node B are $h_{in} = 7.0$ and $h_{out} = 1.3$. Their embedded distance in the 2D Euclidean plane is 5.5. We compute the random walk distance from node A to B (the top dash line in Figure 2.14), $d(A, B)$, by summing node A's h_{out}, the 2D Euclidean distance and node B's h_{in}, which is 18.7 in total. Similarly, the distance from node B to node A (the bottom dash line) is the sum of the Euclidean distance and node B's h_{out} and node A's h_{in}, producing a distance of 14.9.

Embedding Process. By treating the node heights as two extra components in the coordinate system, we use this new space into the embedding process proposed in Section 2.3. The resulting new graph coordinate system is called as *Leo*. The main process is driven by optimizing the coordinate positions and height vectors to minimize distortion between the embedding and the ground truth of the graph. The key change is that we must compute ground truth in terms of random walk distances, and use Equation (2.4) for node distance.

2.5.3 Fast Precomputation

As we mentioned earlier, a critical challenge in embedding random walk distances is the cost of ground truth computation. When applying the Orion embedding process, the embedding must make $(2 \cdot L \cdot n) \cdot 2000$ pairwise random walks to measure actual hitting time (and commute time) between landmarks and non-landmark nodes. Here L and n are the number of landmarks and graph nodes, respectively. For a large graph like the Egypt Facebook graph, it takes around 60

days just to compute the ground truth distances between a single landmark node and all other nodes.

To address this challenge, we propose a novel precomputation method that reduces the pairwise random walks to $(L+n)\cdot 2000$, reducing the total embedding time for Egypt from 60 to only 7 days.

Multi-destination Random Walk. The fast precomputation algorithm is based on a random walk with multiple destinations. Running such random walk from a node can produce random walk steps from this node to multiple nodes, which is similar to BFS algorithm. More specifically, a random walk starting from node i follows its definition to select its next step. If the random walk visits a node j for the first time, we record the current walk steps as one trial for hitting time measurement from node i to node j. Instead of stopping this random walk as defined hitting time, the random walk continues and records its current steps when it reaches a new node for the first time. This random walk can stop when it visits require number of k nodes or its steps get to the maximum steps. As a result, one such random walk can measure the steps to multiple nodes when they are visited for the first time.

Although such random walk with multiple destinations can reduce the number of random walks a lot, it is still not scalable to large graphs if it stops when it visits all graph nodes. Take Egypt as an example again. One such random walk takes 1.5 minutes from a node to reach each node in the graph. To get the converged hitting time from a node, we need to repeat such random walk for 2000 times, which takes 48 hours. To understand this efficiency of random walk, we plot the percentage of nodes visited by the random walk vs. the computation time

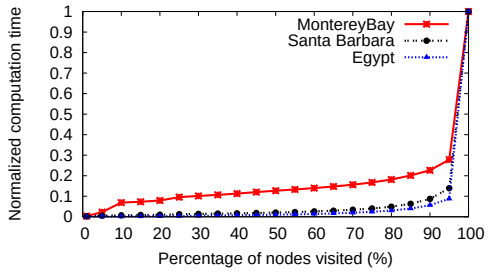


Figure 2.15: Percentage of visited nodes vs. the computation time of a random walk normalized by the time to visit all nodes.

normalized by the time visiting all nodes in Figure 2.15. We find that 90% of the nodes can be visited within $10\% \sim 20\%$ of the normalized computation time. That means in a random walk visiting all nodes, more than 80% of time is used to visit less than 10% of nodes, which is the main reason causing high computation cost. Thus, we explore a tradeoff between efficiency and quantity of visited nodes and find that it is a better compromise when a random walk visits 90% of nodes.

We run the random walk starting from a node for more times to make up the 10% not visited nodes. Recall that we measure stable hitting time from node i to node j by repeating the random walk from node i to j for 2000 times. Similarly, we have to repeat the random walk with multiple destinations for several times for a reasonable expectation. In addition, since such random walks have no explicit destinations, we cannot promise each visited node can be visited for 2000 times, which can provide a stable hitting time, after 2000 times repeating random walks. In other words, we may need to run such random walks for more than 2000 times. We empirically repeat the random walk starting from a node for N times, where N is from 2000 times to 6000 times. In Figure 2.16, we show the percentage of nodes which are visited for at least 2000 times by random walks when we repeat random walks for different times. We find that when we repeat random walks for more

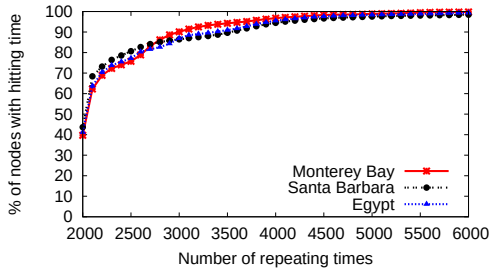


Figure 2.16: Percentage of nodes with stable hitting time vs. Repeating times of random walks

times, the percentage of nodes with stable hitting time increases. Specifically, when we repeat the random walk for 2000 times, only 40% of the nodes get stable hitting time. When the repeating time is 6000 times, 99% of the nodes have stable hitting time. Thus, starting from a node, we repeat the random walks for 6000 times to get more nodes with stable hitting time.

For the remaining 1% nodes without stable hitting time, we run simple end-to-end random walks from the source node to it to ensure that they are visited for 2000 times. Finally, we can compute the ground truth of hitting time from one node to all the other nodes in the graph more efficiently.

In one word, this optimized algorithm works based on random walks with a soft cutoff. That is, a random walk starting from node i records the steps to each node that is visited for the first time and stops when it visits 90% of nodes in the graph. To get stable hitting time for each visited node, we repeat such random walk for 6000 times. For the remaining nodes which are visited for only $k(\leq 2000)$ times, we carry out end-to-end random walks from node i to it for $2000 - k$ times to guarantee that it has stable hitting time.

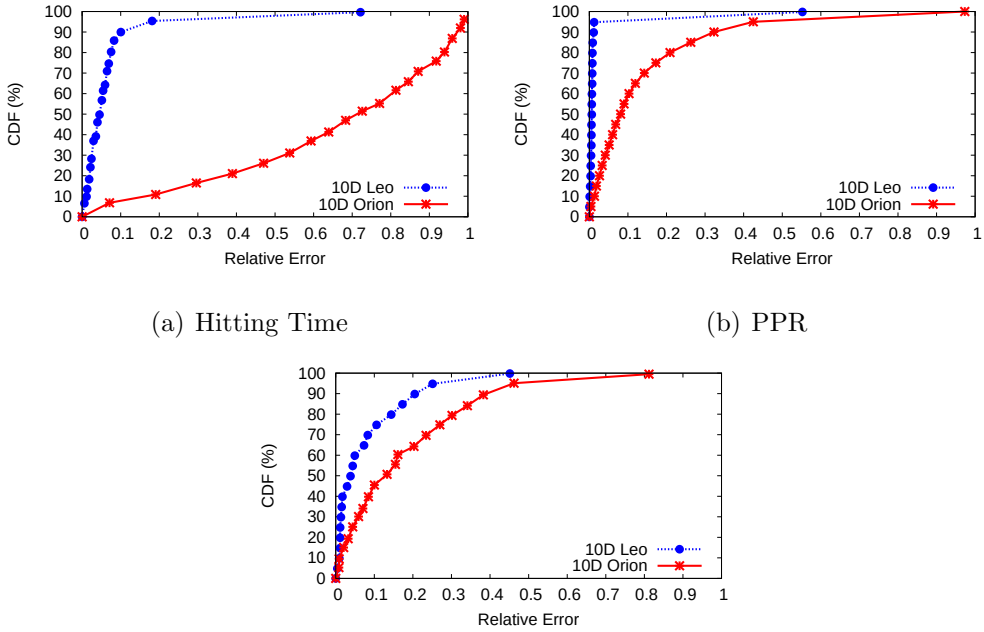
We can directly run this algorithm for each landmark to fast compute the actual hitting time from the landmark to all the nodes. To compute the ground

truth from non-landmarks to landmarks, we can use this algorithm for each non-landmark with a small modification. In detail, since each node only needs to compute the hitting time to a subset of landmarks, *i.e.* 16 landmarks, the random walk starting from it stops when 16 landmarks are reached. As a whole, we can efficiently measure the ground truth of hitting time and commute time for the embedding process using this fast computation algorithm.

2.5.4 Performance Evaluation

In this section, we first understand the performance of Leo in terms of accuracy and speed. Specifically, we investigate the accuracy of random-walk distance estimation using Leo compared against Orion and study the impact of number of dimensions on its estimation accuracy. Then, we measure its efficiency by using average response time for pairwise queries. Second, we examine the utility of this system in two important applications built on random-walk distances, *i.e* search ranking and link prediction.

Accuracy *Compared to Orion.* We examine the accuracy of hitting time, commute time and PPR embedding on the seven graphs in Table 2.1. To make fair comparison to the accuracy of 10-dimension Orion, we use Leo to embed all three distances into a space of 10-dimension Euclidean coordinates plus 2 heights. For simplicity, we call the space of d-dimension Euclidean coordinates plus 2 heights in Leo as a d-dimension Leo. Since hitting time computation is intractable, we sample 1000 random pairs of nodes from each graph and measure the actual hitting time, commute time and PPR between them for comparison. To avoid possible impact of landmarks on results, we choose the 1000 node pairs randomly from all



(a) Hitting Time

(b) PPR

(c) Commute Time

Figure 2.17: Relative error CDF of embedding Hitting Time, PPR and Commute Time in Egypt using 10D Leo vs. 10D Orion.

non-landmark nodes. We use two metrics to quantify the accuracy of the embedding system. One is relative error, and the other is the 90^{th} percentile relative error over all nodes pairs (90% *relative error*). Since Leo performs consistently better than Orion, we focus on the CDF of relative error in Egypt and show the 90% relative error of all graphs.

Figure 2.17(a) shows CDF of relative error of hitting time estimation using Leo compared to the results from Orion. We find that Leo can significantly improve the estimation accuracy for hitting time. Specifically, for 90% of node pairs, the relative error is less than 0.1 using Leo while the relative error of Orion is more than 0.95. In other word, the accuracy improvement of Leo is 90%.

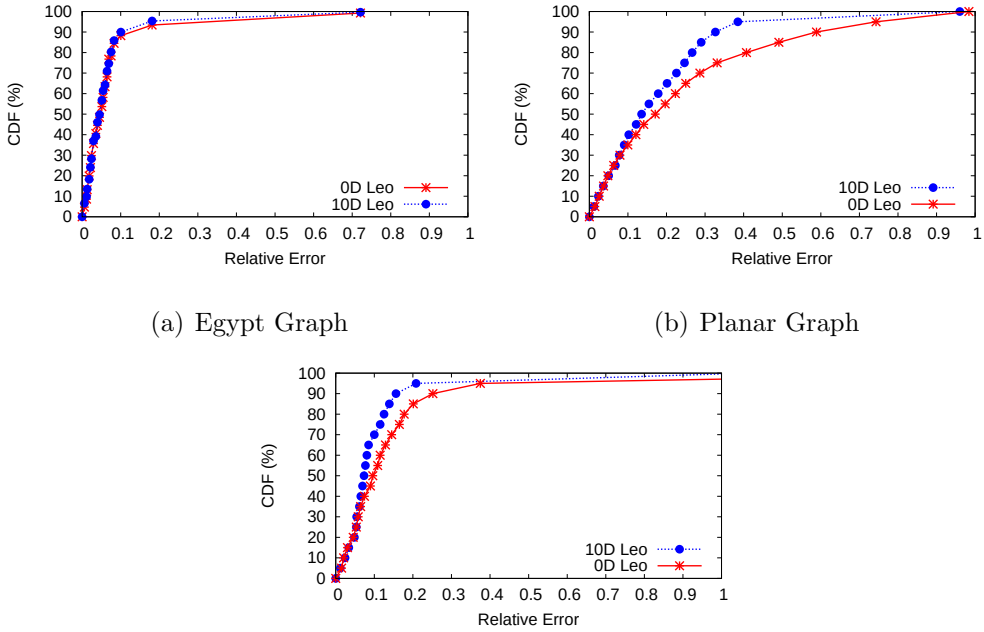
Metric	System	MontereyBay	SB	Egypt	Collab	AS	Citation	P2P	Email	Amazon	Web	Planar
Hitting Time	Orion	0.970	0.919	0.977	0.988	0.971	0.962	0.914	0.987	0.945	0.956	0.969
	Leo	0.119	0.058	0.100	0.121	0.084	0.058	0.100	0.161	0.238	0.058	0.322
PPR	Orion	0.307	0.318	0.323	0.314	0.331	0.300	0.342	0.351	0.342	0.322	0.354
	Leo	0.004	0.006	0.003	0.004	0.004	0.005	0.006	0.003	0.004	0.005	0.006
Commute Time	Orion	0.311	0.333	0.383	0.352	0.325	0.341	0.301	0.311	0.362	0.381	0.375
	Leo	0.211	0.244	0.214	0.242	0.232	0.199	0.205	0.211	0.223	0.214	0.283

Table 2.7: 90th percentile relative errors for Hitting time, PPR and Commute time (Leo vs. Orion w/ 10 dimensions)

We plot CDF of relative error of PPR estimation in Figure 2.17(b). Similar to results in Figure 2.17(a), it shows that Leo is more accurate in estimating PPR. For example, for 90% of node pairs in Figure 2.17(b), the relative error of Leo is 0.005. This is much smaller than the relative error of Orion, *i.e.* $0.3 \sim 0.4$ for 90% pairs of nodes. The accuracy in estimating PPR is improved 98% by Leo. Both the results in Figure 2.17(a) and 2.17(b) show that the two heights introduced in Leo can accurately capture the asymmetric random-walk distances.

We also use Leo to embed symmetric commute time and compare its accuracy to the results of Orion in Figure 2.17(c). It shows that Leo significantly outperforms Orion that was designed for symmetric distances. Still for 90% pairs of nodes, Leo produces relative error 0.2 while the error in Orion is 0.38. Our measurement shows that low degree nodes tend to have large heights while high degree nodes tend to have small heights. This means that our heights can help to capture the local structure around nodes that have poor connection to the core of the graph. Thus, Leo can also capture symmetric distances more accurately than Orion.

We show the 90% relative error of 10-dimension Leo and 10-dimension Orion over all graphs in Table 2.7. Leo is consistently more accurate than Orion across all graphs and all metrics. For hitting time, the accuracy is improved by $67\% - 94\%$



(a) Egypt Graph

(b) Planar Graph

(c) Email Graph

Figure 2.18: Impact of embedding dimension on the accuracy of hitting time.

by Leo. Among all graphs, we notice that the 90% relative errors of three sparse graphs, *i.e.* Planar, Email and Amazon, are slightly higher. Their much lower density means random walks pay a lower price both exiting their local cliques and trying to find their destination nodes. Not surprisingly, our results show that the symmetric global structures make up a much bigger component of the total random walk distance in these graphs. The higher relative errors likely come from estimation errors in the symmetric global distances. For PPR, we find that Leo improves the accuracy $98\% - 99\%$ for all graphs. For symmetric commute time, Leo also produces consistently better accuracy for all graphs, by $25\% - 44\%$ compared to Orion.

Impact of Dimensionality. We also study how the number of dimensions used in the embedding impacts the accuracy of random-walk distance estimation. We vary the dimensions of the Euclidean coordinates from 0 to 10. A 0-dimension space in our system means that no Euclidean coordinates are used for embedding the global structure component, and we only use two heights to represent a node's local structure. Since it is symmetric, the accuracy of commute time embedding increases as the embedding dimension increases. In addition, the number of dimensions has no significant impact on PPR accuracy. Thus we omit those plots for brevity and instead focus on the hitting time results. For clarity, we show the embedding accuracy using 0-dimension Leo and 10-dimension Leo. Figure 2.18(a) shows the results of hitting time in Egypt to demonstrate the impact. We find that the accuracy of 0-dimension Leo embedding is similar to than 10-dimension Leo embedding. This indicates that the asymmetric local structure of Egypt graph dominates the total random walk distance. In other words, two heights are enough to accurately capture the hitting time in the Egypt graph. In fact, we found this to be consistently true for our small-world and high density graphs, including the other two social graphs.

However, the results in our sparser, more hierarchical graphs look quite different. These include the Planar, Email and Amazon graphs. Figure 2.18(b) shows that 10-dimension Leo embedding of planar graph is more accurate than its 0-dimension Leo embedding. For example, for 90% nodes, the relative error in 10-dimension embedding space is 0.3, which is half of the error in 0-dimension space. We observe the same trend in the other three graphs and show the results of Email graph in Figure 2.18(c). Both results show that the symmetric component in the design of Leo, *i.e.* the Euclidean coordinate in Equation 2.4, is necessary,

Metric	Graphs	One-thread Bootstrap (hours)		Parallel Bootstrap (hours)		Per-query response (ms)		
		Precomputation	Embedding	Precomputation	Embedding	Ground Truth	Orion	Leo
Hitting Time	Egypt	168.12	1.59	1.88	0.12	566,359	0.0089	0.0089
	Amazon	157.23	1.93	1.67	0.13	162.08	0.0080	0.0085
	Web	235.12	2.11	2.39	0.15	1463.99	0.0084	0.0081
PPR	Egypt	11.20	1.63	0.16	0.15	17.5	0.0082	0.0087
	Amazon	12.09	1.94	0.17	0.14	18.2	0.0085	0.0088
	Web	12.15	2.12	0.17	0.15	17.9	0.0087	0.0085
Commute Time	Egypt	168.12	1.60	1.88	0.12	1,132,719	0.0082	0.0081
	Amazon	157.23	2.03	1.67	0.12	345.11	0.0082	0.0080
	Web	235.12	2.09	2.39	0.14	3,012	0.0083	0.0082

Table 2.8: Computation time of Leo on three largest real graphs, including bootstrap time and per-query response time.

especially as the symmetric global structure of network increases. Again, this validates our hypothesis that the less dense a network is, the lower the cost of exiting local subgraphs and finding destination nodes. Thus the relative cost of our directional height vectors decreases, and the symmetric component grows in importance.

Embedding and Query Performance. We study the efficiency of our embedding system in this section, including up-front bootstrap costs and average response time for a query. To evaluate the bootstrap time, we measure results for both a single thread instance and a distributed version parallelized across 100 servers. All experiments are measured on a 2Ghz, 8-core Intel Xeon machine with 192GB RAM, and all graphs are embedded into a 10-dimension Leo using 2 height vectors per-node. We show computation time on Egypt, Amazon and Web graphs.

The bootstrap process of Leo includes two phases. The first phase is the precomputation phase, which is to compute the actual distances between landmarks and non-landmarks. We apply our proposed fast precomputation algorithm to compute hitting time and commute time in this phase. The second phase is to

embed the random walk distances into a low-dimension space. We measure the computation time of the precomputation phase and embedding the graph into a 10-dimension space using one single thread. To minimize the bootstrap time, we then parallelize the bootstrap across 100 servers and use the longest computation time of the 100 servers as the parallel bootstrap time.

We show the bootstrap time for hitting time, commute time and PPR using one single thread in Table 2.8. Since the commute time between node i and node j is the sum of hitting time from node i to j and hitting time for the reverse direction, the computation time of commute time between landmarks and non landmarks is equal to the time to compute the ground truth of hitting time. Since the maximum random walk hops for PPR is $log(n)/\alpha$, much smaller than the network size, its computation time is much faster than hitting time.For each graph, we find that the majority of bootstrap time is used to measure the ground truth between landmarks and non-landmarks. For example, to embed hitting time in Egypt, the precomputation for hitting time requires 168 hours while the embedding time is around 2 hours using one single thread.

We parallelize the bootstrap across 100 servers to reduce the bootstrap time, and results are shown in Table 2.8. Since the precomputation process is embarrassingly parallel, we do achieve very high speedups (90x for hitting time and commute time, and 70x for PPR). Since landmarks can only be embedded by one single thread, the parallel embedding time is the sum of landmark embedding time and non-landmark embedding time on the slowest server. Table 2.8 shows that parallelizing embedding reduce the time taken from around 2 hours to less than 10 minutes for our largest graphs. Next, we measure the average per-query response time for Leo, Orion, and the traditional Monte Carlo measurement method. Aver-

age response time is defined as the average time to compute the expected or stable random walk distance for a node pair. We average the computation time across 1000 random pairs of nodes. Table 2.8 shows the average response time using Leo, Orion, and the average response time to compute ground truth using the traditional method. As expected, response time on Leo is constant for different graph sizes, and is several orders of magnitude faster than traditional methods. For example, the time to estimate hitting time using Leo is 0.008ms ($8\mu s$), 8 orders of magnitude faster than the time required to compute the ground truth, $\sim$ 10 minutes. As expected, Table 2.8 also confirms that per-query response times on Leo and Orion are essentially identical. Once the upfront bootstrap phase is complete, Leo's response time is 8 microseconds for hitting time, PPR, and commute time queries, which makes it more than capable of handling real-time queries on large graphs.

Applications. Now, we evaluate the utility of graph coordinate systems at the application level. We apply our embedding system into two popular applications, search ranking and link prediction. The two applications are built on expensive random walk distances, but are useful in practical search engines and social network analysis. Our experiments show that using graph coordinate systems in these applications can produce answers that closely approximate results derived from measured (ground-truth) random walk distances[3].

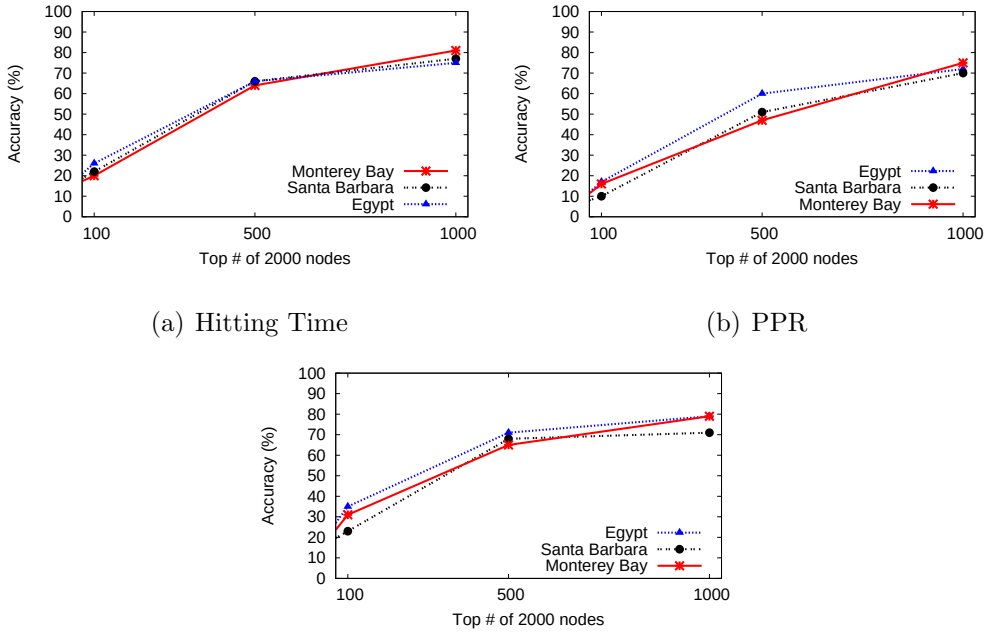
Search ranking. Ranking search results or entities based on their relevance is a fundamental problem in search engines [32] and recommendation systems [31]. For example, Google might return thousands of answers for the query "publications

[3]We use Leo with 10 dimensions.

about social network published in 2012." For a better user experience, Google ranks the returned results based on the relevance metrics such that the most wanted results by the user should be prioritized. Among the metrics to quantify relevance, random walk distance is one of the most important and widely used metrics [32, 31, 30].

We implement a search ranking application to evaluate the impact of Leo. We choose a random node i to send out a query for which N answers are returned. Here, each answer is represented by a node in the graph. We then measure the random walk distances from node i to each node j in the set of N nodes. Finally, we rank the N nodes based their distances to node i and select the top K nodes as the best results for the query. We separately use commute time, hitting time and PPR as random walk distances in the ranking. When using commute or hitting time, we rank the result in an increasing order such that results with low commute time or hitting time are in top positions. In contrast, using PPR, we rank the results in descending order and the top K nodes have the highest PPR.

In our experiment, when a node sends out a query, $N = 2000$ random nodes return answers. We rank them using their distances to the query origin. Finally, we return the top $K = 100, 500, 1000$ answers, which is corresponding the top $5\% \sim 50\%$ answers. We repeat this experiment 2000 times. Each time we choose a random node to generate a query and rank the nodes using commute time, hitting time and PPR independently. For each random walk distance, we get two sets of top K nodes. One set is generated using measured actual distance and the other set is based on the distances estimated by Leo. We count the amount of overlap between the top K nodes in the two sets. We use the ratio of the number of overlapping nodes to the total number of top K nodes to quantify the accuracy.



(a) Hitting Time

(b) PPR

(c) Commute Time

Figure 2.19: Accuracy of Top k ranked nodes.

All our experiment is measured on the three Facebook graphs in Table 2.1, *i.e.* MontereyBay, Santa Barbara and Egypt.

We plot the results of hitting time, commute time and PPR in Figure 2.19. It shows that for all three distances, our system can more accurately approximate the ground truth as K is larger. For example, using commute time in Egypt graph, the accuracy increases from 70% to more than 80% when K increases from 500 to 1000.

Link prediction. Social network is a network with high dynamics. The addition of edges is one of the main reasons to cause the frequent changes in network. Predicting link creation in the future is one important problem for understanding network evolution and predicting network growth. [107] shows that node structure

Graphs	Hitting Time (%)		PPR (%)		Commute Time (%)	
	GT	Leo	GT	Leo	GT	Leo
MontereyBay	71.21	69.12	72.16	69.56	61.23	69.11
Santa Barbara	69.28	71.28	71.22	70.01	65.26	70.26
Egypt	73.52	69.93	71.56	72.38	68.01	67.98

Table 2.9: Link prediction application accuracy using hitting time, commute time, and PPR, based on Ground Truth (GT) and Leo.

similarity in a network can be used to predict links. This paper uses several metrics to quantify nodes similarity. Among them, commute time, hitting time and PPR are three important metrics with high accuracy. However, the computation of these similarity metrics is very expensive. This motivates several studies [151, 156] to accelerate the random walk distance estimation and is one important application to evaluate the accuracy of the estimation. Thus, we use our system to estimate random walk distances to predict future links and compare the predicting accuracy to the accuracy generated by the actual distances. Again, we separately use hitting time, commute time and PPR in link prediction.

Our link prediction experiment is similar to [151]. We delete 10% random edges from each graph G for prediction, which results in a new graph G'. We test how accuracy of our system is in predicting the deleted 10% edges. Since measuring the actual distances for all nodes pairs as ground truth is costly in terms of computation time, we only consider all pairs of nodes that have edges deleted as potential edges. Then we rank all potential edges based distances between their two endpoints in graph G'. Similar to search ranking, we rank commute time and hitting time in an increasing order while rank PPR in a descending order. We choose top M pairs of nodes in the ranked edge list, where M is the exact number of the 10% deleted edges in the graph. Again, we can have two sets of top M

results using actual measured distances and estimated distances from Leo for each distance metric. For each set, we count how many of edges, *i.e.* node pairs, in this set overlap with the actual deleted edges and use the ratio of the overlapping edges to the total M edges as the accuracy metric. Thus, we can compare the prediction accuracy using Leo to the prediction accuracy using actual distances.

We run our experiment on the three Facebook graphs and show the results of the three random-walk distances in Table 2.9. For each metric, we find that the prediction accuracy of Leo is quite similar to the accuracy using actual distances. For example, in Egypt, using hitting time in link prediction, the accuracy using actual distances is 73.52% while the accuracy of Leo is 69.93%. We notice that Leo outperforms actual distances in few cases. This is because the estimation error of Leo results in that some deleted edges are ranked higher than they should be. However, the number of this kind of node pairs is relatively small. The prediction accuracy of Leo is almost as accurate as the results using actual distances.

2.6 Related Work

In this section, we briefly summarize other related work, including shortest path estimation methods, random walk distance estimation, studies on network coordinate systems and applications using node distances.

2.6.1 Shortest Path Estimation

Since exact shortest path computation methods, such as BFS, Dijsktra and Fast shortest path distance estimation in large networks, fail to scale with graph size, several fast algorithms [145, 147, 141, 42, 67] are proposed to efficiently

estimate shortest path, which can be classified into two classes. The first class of algorithms is to apply embedding methods to shortest path queries. [145] is an initial work to embed metrics in small graphs into a Euclidean space. [147] proposes a network structure index (NSI) to compute node positions in a graph. [141] is a landmark scheme for approximating shortest path distances by storing for each node its distance to every landmark. All three algorithms are with significant limitations in scalability.

The second class is Sketch-based algorithms [42, 67], which precompute and store BFS trees rooted in each landmark. For any pair of two nodes, the algorithms use the results of BFS trees to approximate their shortest path length and locate the path. Comparing with different sketch-based algorithms, the graph coordinate systems, *e.g.* Rigel, are more efficient with similar level accuracy.

2.6.2 Random Walk Distance Estimation

Given the prevalence of Personal PageRank in search engines and recommendation systems, researchers have developed two general approaches to compute PPR, *i.e.* linear algebraic optimization [80, 119] and Monte Carlo approximation algorithms [52, 11]. Monte Carlo algorithms to compute PPR can be significantly sped up using a variety of techniques, ranging from parallelization via MapReduce [16] to improved bounds for distributed algorithms [43].

Since commute time is symmetric, many have tried to approximate it using fast matrix computation. Standard matrix computations based on matrix inverse require $O(n^3)$ time, which does not scale. One solution is to produce fast approximations using the Lanczos process [30]. Since network effective resistance is

analogous to commute time, [157] uses graph sparsification to compute effective resistances between any pair of nodes in $O(\log n)$ time. Finally, [151] focuses on efficiently computing hitting time and commute time within a fixed number of T hops, instead of a generalized query between any two nodes.

2.6.3 Network Coordinate Systems

Embedding techniques have been used in a variety of application contexts. The most recent and well-known use of such techniques was in the context of network coordinate systems [133, 54, 41, 37], which are efficient and scalable mechanisms to estimate Internet latencies without performing end-to-end measurement. In contrast, graph coordinate systems are designed to preserve node distances in large complex graphs.

We summarize the studies on network coordinate systems with the three popular geometric spaces, *i.e.* Euclidean, Spherical and Hyperbolic. A Euclidean space is widely used to predict routing latency between hosts [133, 41, 37, 162, 152]. For example, GNP [133] is a centralized system that uses a small number of public landmarks to embed all Internet hosts in the space. Similar systems proposed later include those using Simplex Downhill [128] to optimize host coordinates [37], Lipschitz embedding [162], a spring force model [41], and most recently a system using Euclidean Big-Bang Simulation [152]. These systems calibrate nodes' geometric positions based on distances, *e.g.* Internet round-trip time (RTT), which are measured in a distributed manner. Still later work proposed bounds on the distortion of Euclidean embedding. To the best of our knowledge, J. R. Lee's re-

cent result [96] proves the tightest upper bound, $O(\sqrt{\log n} \log\log n)$ for an n-point Euclidean embedding.

Spherical embedding was first studied in Vivaldi [41]. While morphing on spherical spaces is widely used in computer vision [89], there is little theoretical work investigating spherical embedding.

Intuitively, a hyperbolic space can better model Internet topology with tightly connected cores. Thus, several experimental systems for embedding Internet distances [113, 153, 114] use a hyperbolic space to improve the embedding accuracy. In the context of ad hoc networks, a *greedy hyperbolic embedding* in [88] yields routes with low stretch, where greedy embedding is a graph embedding with the following property: for any pair of nodes (u, v), there is at least one neighbor of node u closer to node v than node u itself. A later work [40] improves the greedy embedding algorithm for dynamic graphs, and proposes a modified greedy routing algorithm for message routing. They either focus on graphs in the context of routing in wireless networks or on small synthetic graphs ($\sim$50 nodes as in [40]). A recent project [138] proposes a graph model using hyperbolic spaces that is capable of producing synthetic graphs with scale-free structural properties. Unlike our work, this project aims to generate synthetic graphs instead of embedding real graphs.

In addition to the three geometric spaces, Vivaldi [41] augments a Euclidean coordinate with a height. This is used to capture the congestion delay from a node to cores of Internet, which is for symmetric routing latencies. In contrast, the directional height space for random walk distances decouples incoming and outgoing heights, which successfully adapts to a wide range of graph structures. As

a significantly more general model, the directional height space produces accurate results for both symmetric and asymmetric node distances.

2.6.4 Applications Using Node Distances

There are many social applications based on shortest path length. For example, distance-ranked social search ranks search results by the proximity of each result to the user in social graphs [141, 123]. Information dissemination [33] can leverage distances between nodes to find the most influential nodes. Community detection algorithms on social graphs (see taxonomy from [53]) can benefit from shortest path distances between nodes to classify them in different clusters. Furthermore, Sybil attack detections are in essence based on community detection strategies [171], which make them suitable candidates to leverage our system. Neighborhood function [137] uses node distance distribution to predict whether two graphs are similar or not. Mutual friends detection computes the mutual friends between social users. In [159], the auction site calculates social distances to identify items auctioned within social circle defined by users. All these applications are based on shortest path computations, therefore, in essence, they can benefit from our system.

Random walks appear in numerous applications in fields such as computer vision, data mining, network security and social network analysis. For example, work in computer vision [63] reliably extracts shape properties in Silhouettes using hitting time. [64] utilizes hitting time from all nodes to a chosen node as threshold to determine automated graph partitions. Commute time is an important way to track real-world multi-body motions [144] and image segmentations [143]. In

data mining problems, standard clustering algorithms such as K-means produce more accurate results by replacing traditional distance with commute time [181]. Personal page rank has been used to improve partitioning in [9], and commute time has been used to detect global and local outliers in data [85]. Since random walks are resistant to noise and manipulation, they are also widely used to build robust reputation systems [74] and Sybil detection systems on social networks [183, 182]. In social networks, commute time, hitting time and PPR are important metrics to accurately perform link prediction over time [107].

2.7 Summary

Node distance computation, including shortest path length and random walk distances, is one of the most critical primitives for both graph analysis and applications. Unfortunately, traditional algorithms for node distance computation no longer scale with big real graphs with millions of nodes and billions of edges. In this chapter, we explore a novel technique, graph coordinate systems, to accurately approximate node distances in constant time.

To estimate shortest path distances, we propose Rigel, a hyperbolic graph coordinate system. We discuss the impact of geometric spaces on the estimation accuracy, and show that the hyperbolic space can better model large complex networks in terms of the accuracy. To scalably embed large graphs, we naturally parallelize the embedding process across multiple servers. For large graphs like Renren with 43 million nodes and 1 billion edges, Rigel not only produces more accurate results than using other geometric spaces, but also replies node distance queries 5 orders of magnitude faster than BFS. In addition, we propose Rigel path,

a heuristic shortest path finding algorithm using the generated coordinates. The measurements on various big real graphs show that Rigel path matches the best accuracy produced by prior work while returning results up to 18 times faster than state-of-the-art shortest-path systems with similar levels of accuracy.

To account for the asymmetry of random walk distances, we propose two "height vectors" to model per-direction, per-node random walk costs. In an abstract sense, one height captures the cost of leaving the subgraph around the source node, and the other one captures the cost of finding the destination in the local subgraph. We show that these factors change dramatically for a variety of graph topologies. Particularly, for small-world graphs, asymmetric hitting time is dominated by a single per-destination cost. The results show that Leo, the graph coordinate system based on the directional height space, accurately predicts both symmetric and asymmetric random walk distances while responding the queries in microseconds, 8 orders of magnitude faster than existing methods.

Chapter 3

Analyzing and Modeling Dynamics in Big Real Graphs

3.1 Introduction

[1]As the emergence of massive Online Social Networks (OSNs), a deeper understanding of the dynamics in these networks have numerous practical implications on many social network specific applications, including the design of infrastructure, applications, and security mechanisms for social networks.

However, despite recent progress in the areas of analyzing and modeling OSNs [21, 57, 78, 101, 124, 58], their network dynamics is still poorly understood. Although there is general agreement that OSNs are structures that are highly dynamic in nature and driven by a number of interrelated dynamic processes, most current works tend to study them only via *static* snapshots [21, 57, 78, 124, 58], or seek to capture network dynamics as a single process [101, 124, 58], such as preferential attachment (PA). As a result, current models of network dynamics [5, 6, 84, 104]

[1]Abbreviated version of content in Section 3.2 and Section 3.3 can be found in paper "Multi-scale dynamics in a massive online social network" [185]. The content in Section 3.4 and Section 3.5 does not yet appear in any currently published paper.

focus primarily on a final graph with some desired structural properties, but fail to model or match the sequence of dynamic events that leads to the structure.

Our goal in this chapter is to build a model of social network dynamics that successfully reproduces not only time-dependent structural properties of the network, but also the sequence of dynamic events leading to the structure and its evolution in time. Such a detailed dynamic graph model would address a number of practical OSN problems. First, the research community has repeatedly expressed a need for real dynamic graph traces. Using a real trace for calibration, our model can generate "realistic" dynamic graphs with a complete list of time-stamped network events. Second, our model can be used to perform "interpolation", *i.e.* constructing complete dynamic graph traces that connect static snapshots of OSNs. Given successive static snapshots from OSNs, our model can approximate the continuous network evolution between them. Finally, our model can be used to detect abnormal events in real networks, *i.e.* events that disrupt the expected or "normal" network dynamics. Such events might represent malicious attacks or significant changes in user behavior.

To achieve the goal, we understand in detail the evolutionary dynamics in a social network in term of both structural dynamics and temporal dynamics. Specifically, to better study the evolution of network structure, we measure dynamic social network graphs at three network scales, including nodes, communities, and networks. In contrast with prior studies in network structural dynamics [18, 101, 124, 58], which modeled or validated the network dynamics as a single process, the multi-scale measurement method can help us to learn the interrelated dynamics processes and how they impact users' activities. To understand temporal properties in network growth, our approach is heavily influenced by past

work on network traffic modeling that showed measured network traffic to exhibit a statistical property called *self-similarity*, which is different from popular traffic models such as the well-known Poisson processes. Similarly, whether the observed dynamics of social networks do in fact exhibit self-similarity will have a significant impact on the way we view and consider dynamic graph models. Based on the observations, we develop a model capturing both spatial and temporal properties.

In this chapter, our work focuses on analyzing and modeling a large dynamic online social graph, *i.e.* Renren. With over 220 million users, Renren is the largest social network in China, and provides functionality similar to Facebook. The anonymized Renren data studied in this chapter includes timestamps of all the first two-year events, including the creation of 19 million user accounts and 199 million edges. This captures the network's initial burst of growth, as well as a period of more sustained growth and evolution. This dataset is notable because of three features: its scale, the absolute time associated with each event, and a rare network *merge* event, when the network merged with its largest competitor in December 2006, effectively doubling its size from 600K users to 1.3 million users in a single day.

Throughout this chapter, we make three key contributions. First, we analyze the Renren network at three network scales, including nodes, communities and networks. Our analysis produces a number of interesting findings of dynamics at different scales. First, at the level of individual nodes, we find that new edge creation is increasingly dominated by existing nodes in the system, even though new node arrivals is keeping pace with network growth as the network matures. At the same time, the influence of the preferential attachment model weakens over time. Second, at the level of user communities, we find that users in large

communities are more active in creating friends. Active nodes with high degrees tend to join and help form large communities, and their activity introduces new friends to their neighbors, further encouraging edge formation within the community. Finally, in our analysis of the network merge event, we use user activity to identify duplicate accounts across the networks. We also find that the network merge event has a distinct short-term impact on user activity patterns.

Second, we detect and quantify the self-similarity in the edge creation process at various time scales. We find that edge creation in the Renren online is non-stationary over long term periods, even after removing the impact of node arrivals by sampling edge creation over a fixed user population. On the contrary, by applying the more robust wavelet-based method for examining self-similarity, we find edge creation in the Renren social network does exhibit properties consistent with self-similarity at small time scales (see Section 3.4). The exhibition of self-similarity in edge creation process has significant impact on modeling dynamics of network growth.

Third, we propose a detailed model for graph dynamics that captures both the temporal properties of graph dynamics (self-similar over time scales from minutes to hours) and spatial properties (long term graph distance shrinkage and slow reduction in local clustering). Our validation shows that it produces dynamic traces that match key dynamic properties of the original graph in both temporal and spatial dimensions. Thus, by producing realistic traces of time-stamped network events, our model fills a large void in the research community (see Section 3.5).

3.2 Dataset and Basic Analysis

We begin our study by first describing the dataset, and performing some basic analysis to understand the impact of network dynamics on first order graph metrics. Our data is an anonymized stream of timestamped events shared with us by Renren, whose functionality is similar to those of Facebook, Google+ and Orkut. Our basic measurements in this section set the context for the analysis of more detailed metrics in later sections.

3.2.1 Renren Dynamic Dataset

The first edge in Renren was created on November 21, 2005. The social network was originally built as a communication tool for college students, named *Xiaonei* or "inside school". But Xiaonei expanded beyond schools in November 2007, and changed its name to Renren ("everyone") in 2009.

Our anonymized dataset encompasses the timestamped creation events of all users and edges in the social network. The dataset covers more than 2 years, starting on November 21, 2005 and ending December 31, 2007. In all, the dataset includes the creation times of 19,413,375 nodes and 199,563,976 edges. To perform detailed analysis on the social graph, we produce 771 graphs representing daily static snapshots from the timestamped event stream. Note that in this chapter, we will use the term *node* to mean an OSN *user* and *edge* to mean a friendship link.

On Renren, default user policy limits each user to 1,000 friends. Users may pay a fee in order to increase their friend cap to 2,000. However, prior work by the network has shown that very few users take advantage of such features. We

make the same observation about our dataset: the number of users with >1,000 friends is negligibly small.

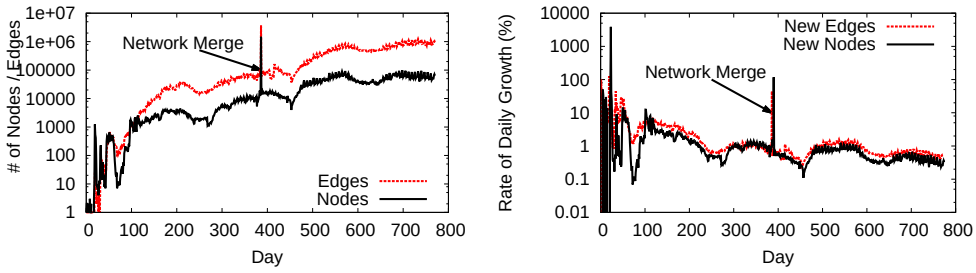
Network Merge Event. An unusual event happened on December 12, 2006, when Renren/Xiaonei merged its social network with 5Q, a competing social network that was created in April 2006. Before the two networks merged, Renren/Xiaonei counted 624K active users and 8.2M edges, and 5Q included 670K active users and 3M edges.

During the merge, both OSNs were "locked" to prevent modification by users, and all information from 5Q was imported and merged into Renren/Xiaonei's databases. Starting the next day, users could log-in to the combined system and send friend requests normally, *e.g.* users with Renren/Xiaonei profiles could friend 5Q users, and vice versa. Since both 5Q and Renren/Xiaonei targeted university students, it was inevitable that some users would have duplicate profiles after the merge. Renren/Xiaonei allowed users to choose which profile they wanted to keep, either Renren/Xiaonei or 5Q, during their first log-in to the site after the merge. New users just joining the system would not notice any difference between Renren/Xiaonei and 5Q user's profiles.

Wherever possible, we treat the merge as an external event to minimize its impact on our analysis. We present detailed analysis of the network merge event in Section 3.3.3.

3.2.2 Network Level Measurement

Network Growth. We begin with measuring the overall network growth. Figure 3.1(a) depicts the growth of the Renren network in terms of the number of



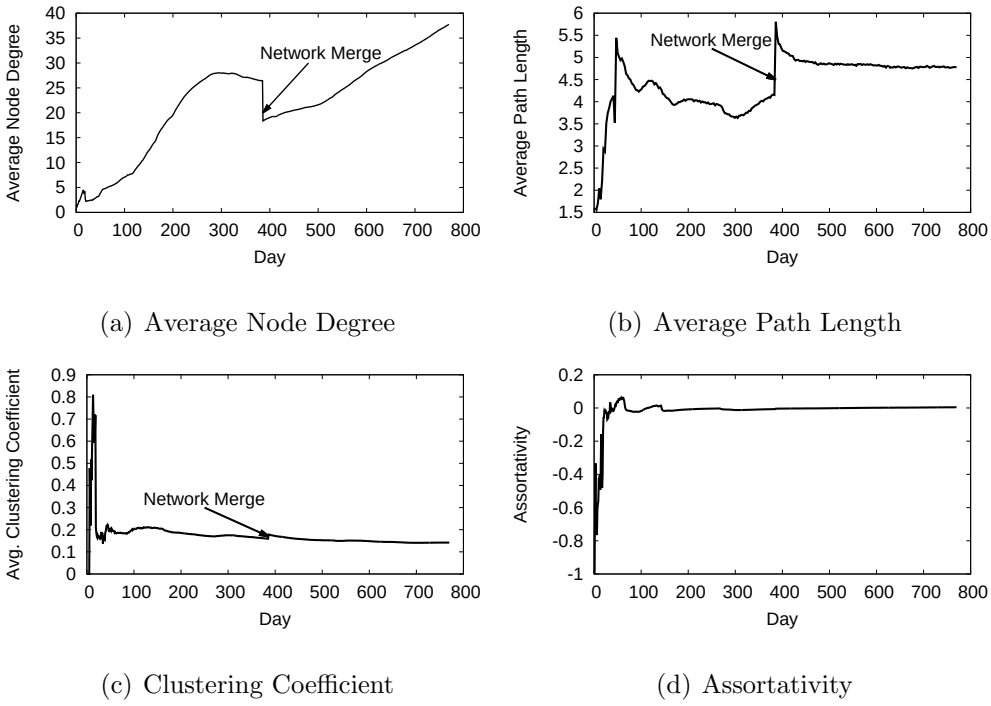
(a) Absolute Network Growth

(b) Relative Network Growth

Figure 3.1: Network growth over time.

nodes and edges added each day. Day 0 is November 21, 2005. Overall, the network grows exponentially, which is expected for a social network. However, there are a number of real world events that temporarily slow the growth, and manifest as visible artifacts in Figure 3.1(a). The two week period starting at day 56 represents the Lunar New Year holiday; a two-month period starting on day 222 accounts for summer vacation; the merge with 5Q network causes a jump in nodes and edges on day 386; additional dips for the lunar new year and summer break are visible starting at days 432 and 587, respectively. In Figure 3.1(b), we plot daily growth as a normalized ratio of network size from the previous day. It shows that relative growth fluctuates wildly when the network is small, but stabilizes as rapid growth begins to keep rough pace with network size.

Graph Metrics Over Time. We now look at how four key graph metrics change over the lifetime of our data stream, and use them to identify structural changes in the Renren network. We monitor average degree, average path length, average clustering coefficient, and assortativity. As before, the analysis of each metric starts from November 21, 2005.



(a) Average Node Degree

(b) Average Path Length

(c) Clustering Coefficient

(d) Assortativity

Figure 3.2: The evolution of four important graph metrics over time.

Average Degree. As shown in Figure 3.2(a), average node degree grows for much of our observed time period, because the creation of edges between nodes out paces the introduction of new users to the network. This trend changes around day 305, when a period of rapid growth in users starts to reduce average degree. This arises from a sudden influx of new users due to several successful publicity campaigns by Renren. In December 2006, average degree drops suddenly when 670K loosely connected 5Q nodes join the Renren network. Average degree resumes steady growth following the event, again showing edge growth out pacing node growth and increasing network densification [104].

Average Path Length. We follow the standard practice of sampling nodes to make path length computation tractable on our large social graphs. We compute

the average path length over a sample of 1000 nodes from the SCC for each snapshot, and limit ourselves to computing the metric once every three days. As seen in Figure 3.2(b), the results are intuitive: path length drops as densification increases (*i.e.* node degree increases). There is a significant jump when 5Q joins Renren on day 386, but resumes a slow drop as densification continues after the merge.

Average Clustering Coefficient. Clustering coefficient is a measure of local density, computed as the ratio of the existing edges between the immediate neighbors of a node over the maximum number of edges possible between them. We plot average clustering coefficient in Figure 3.2(c). In early stages of network growth (before day 60), the network was very small and contained a large number of small groups with loose connections between them. Groups often formed local cliques or near-cliques, resulting in high clustering coefficients across the network. Once the network grows in size, average clustering coefficient transitions to a smooth curve and decreases slowly. The network merge produces a small jump, since the 5Q network had many small clusters of 3 or 4 nodes that boosted average clustering coefficient.

Assortativity. Finally, we plot assortativity in Figure 3.2(d). Assortativity is the probability of a node to connect to other nodes with similar degree, computed as the Pearson correlation coefficient of degrees of all node pairs. In the early stages of the network, the graph is sparse and dominated by a small number of supernodes connecting to many leaf nodes. This produces a strong negative assortativity that fluctuates and then evens out as the network stabilizes in structure. Assortativity evens out at around 0, meaning nodes in Renren have no discernible inclination to be friends with nodes of similar or different degree.

Summary. We observe that the high-level structure of the Renren social network solidifies very quickly. Several key properties stabilize after the first 2 months, with others establishing a consistent trend after 100 days. While the notable network merge with 5Q introduces significant changes to network properties, the effects quickly fade with time and continued user growth.

3.3 Understanding Network Dynamics at Multiple Scales

Our goal of this section is to study in detail the evolutionary dynamics of the Renren network. This includes not only the initial growth process during a social network's formation, but also the ongoing dynamics afterwards, as the network matures. Much of the prior work in this area, including generative graph models and efforts to validate them [18, 101, 124, 58], has focused on capturing network dynamics as a single process. In contrast, we are interested in the question "how are individual user dynamics influenced by processes at different scales?" How much are the dynamics of users influenced by external forces and events, such as the activities of friends in communities they belong to, or by large-scale events that occur at the network level?

In this section, we explore these questions empirically through a detailed analysis of network dynamics in the Renren dataset at multiple scales: at the individual user level, at the level of user communities, and at the global network level.

3.3.1 Edge Evolution

First, we study the behavior of individual nodes in terms of how they build edges over time. Many studies have shown that nodes build edges following the preferential attachment (PA) model [18, 101, 124, 58]. Specifically, when a new node joins the network and creates edges, it chooses the destination of each edge proportionally to the destination's degree. In other words, nodes with higher degrees are more likely to be selected as the destination of new edges, leading to a "rich get richer" phenomenon.

Using the dynamic Renren network data, we extend the analysis of this model in two new dimensions. First, while PA assume that new nodes are the driving force behind edge creation, we seek to understand how node activities are correlated with node age, *i.e.* the time that a node have been in the network. Second, we are interested in whether, as the network evolves, the predictive ability of the PA model grows or weakens over time.

Node Age and Edge Creation. Since most generative graph models, such as the PA model, use new nodes to drive edge creation, we ask the question "What portion of the new edges created in the network are driven by the arrival of new nodes?" For each day in our dataset, we take each edge created on that day and determine its minimal age, *i.e.* the minimum age of its two endpoints. The distribution of this value shows what portion of new edges are created by new nodes.

We compute and plot this distribution in Figure 3.3. We show the relative contribution by nodes of different ages by plotting three stacked percentages, showing the portion of daily new edges with minimal age $\leq$ 1 day, $\leq$ 10 days,

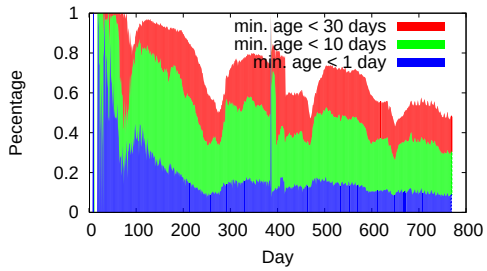


Figure 3.3: The portion of edges created by new nodes each day.

and $\leq$ 30 days. We see that when the network is young ($\leq$ 60 days), the vast majority of new edges connect brand new nodes (*i.e.* 1 day old). As the network stabilizes and matures, that portion quickly drops, and continues to decrease over time. Edges with minimal age of 10-30 days dominate new edges for much of our trace, but their contribution steadily drops over time from 95% around day 100 to 48% by day 770. Note that this drop occurs even after the daily relative network growth has reached a constant level (see Figure 3.1(b)). It is reasonable to assume that in today's Renren network (4.5 years past the end of our data), the vast majority of new edges connect mature users who have been in the network for significant amounts of time.

This result is important, because it shows a dramatic change in the driving force behind edge creation as the network matures. Most generative graph models assume edge creation is driven by new nodes. However, our data indicates that existing models will only accurately capture the early stages of network creation. Capturing the continuous evolution of a mature network requires a model that not only recognizes the contribution of mature nodes in edge creation, but also its continuous change over time.

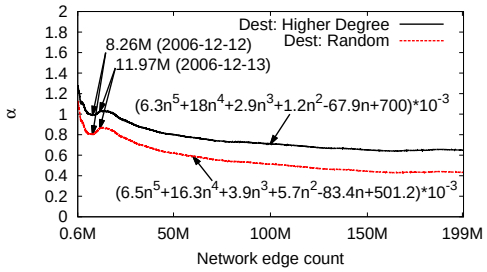


Figure 3.4: Evolution of $\alpha(t)$.

Strength of Preferential Attachment. We then take a look at the preferential attachment model and how well it predicts changes over time and network growth. We follow the method in [101] to measure the strength (or degree) of preferential attachment using edge probability $p_e(d)$. This function defines the probability that an edge chooses its destination with degree d, normalized by the total number of nodes of degree d before this time step:

$$p_e(d) = \frac{\Sigma_t\{e_t(u,v) \wedge d_{t-1}(v) = d\}}{\Sigma_t|v : d_{t-1}(v) = d|} \tag{3.1}$$

where $\{e_t(u,v) \wedge d_{t-1}(v) = d\} = 1$ if the destination v of the edge $e_t(u,v)$ is of degree d, and 0 otherwise.

Intuitively, if a network grows following the PA model, its edge probability $p_e(d)$ should have a linear relationship with d: $p_e(d) \propto d$. The authors of [101] verified this conclusion using synthetic graphs, and also tested the PA model on four real social networks: Flickr, Delicious, Answers, and LinkedIn. The first three networks follow the PA model $p_e(d) \propto d^\alpha$ with $\alpha \approx 1$, while for LinkedIn, $\alpha = 0.6$. From these observations, we can define a criterion for detecting preferential attachment: when $\alpha \to 1$, the network grows with a strong preferential attachment, and when $\alpha \to 0$, the edge creation process becomes increasingly random. Using

this criterion, we validate the level of PA model over time on Renren by fitting $p_e(d)$ measured at time t to $d^{\alpha(t)}$ and examining $\alpha(t)$ over time.

We make some small adjustments to the computation of $p_e(d)$ on the Renren data. First, because our data does not state who initiated each friendship link (edge directionality), we perform our test with two scenarios. The first is biased in favor of preferential attachment because it always selects the higher degree end-point as the destination. In the second scenario the destination is chosen randomly from the two end-points. Second, to make the computation tractable on our large number of graph snapshots, we compute $p_e(d)$ once after every 5000 new edges. Finally, to ensure statistical significance, we start our analysis when the network reaches a reasonable size, *e.g.* 600K edges.

We examine $\alpha(t)$ over time in Figure 3.4. We make two key observations. *First*, $\alpha(t)$ when using the higher-degree method is always larger than when using random selection. This is as expected since the former is biased in favor of preferential attachment. More importantly, the difference between the two results is always 0.2. This means that despite the lack of edge destination information, we can still accurately estimate $p_e(d)$ from these upper and lower bounds.

Second, $\alpha(t)$ decays gradually over time, dropping from 1.25 (when Renren first launched) to 0.65 (two years later at 199M edges). This means that when the network is young, it grows with a strong preferential attachment. However, as the network becomes larger, its edge creation is no longer driven solely by popularity. Perhaps this observation can be explained by the following intuition. When a social network first launches, connecting with "supernodes" is a key factor driving friendship requests. But as the network grows, it becomes harder to locate supernodes inside the massive network and their significance diminishes.

Finally, we observe a small ripple at the early stage of the network growth, when $\alpha(t)$ experiences a surge on December 12, 2006 (8.26M edges). This is due to the Renren/5Q merge event, which generated a burst of new edges that produce a bump in $\alpha(t)$ for that single day.

Summary. Our analysis on the impact of individual nodes on edge creation produces two conclusions:

- *Edge creation in early stages of network growth is driven by new node arrivals, but this trend decreases significantly as the network matures.*
- *While edge creation follows preferential attachment, the strength degrades gradually as the network expands and matures.*

3.3.2 Community Evolution

In online social networks, communities are groups of users who are densely connected with each other because of similar backgrounds, interests or geographic locations. Communities effectively capture "neighborhoods" in the social network. As a result, we believe they represent the best abstraction with which to measure the influence of social neighborhoods on user dynamics. We ask the question, "how do today's social network communities influence their individual members in terms of edge creation dynamics?"

To answer our question, we first introduce the background of community definition and the detection algorithms. We then develop our method to scalably identify and track communities as they form, evolve, and dissolve in a dynamic network. We then present our findings on community dynamics in Renren. Fi-

nally, we analyze community-level dynamics and use our detected communities to quantify the correlation between node and community-level dynamics.

Background. Communities can be defined based on network structure as groups of well-connected nodes. There are dense connections inside communities but sparse connections between communities [132]. Modularity [130] is a widely used metric to quantify how well a network can be clustered into communities. It is defined as the difference between the fraction of edges falling in communities and the expected fraction when edges are randomly connected. It is formally defined in Equation 3.2, where A is the adjacency matrix ($A_{ij} = 1$ if node i and j are connected, and $A_{ij} = 0$ otherwise), k_i is the degree of node i, m is the total number of edges and $\delta(c_i, c_j) = 1$ if node i and j are in the same community and $\delta(c_i, c_j) = 0$ otherwise. The value of modularity should be between -1 and 1, and a large modularity means the network can be well clustered into communities.

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j) \tag{3.2}$$

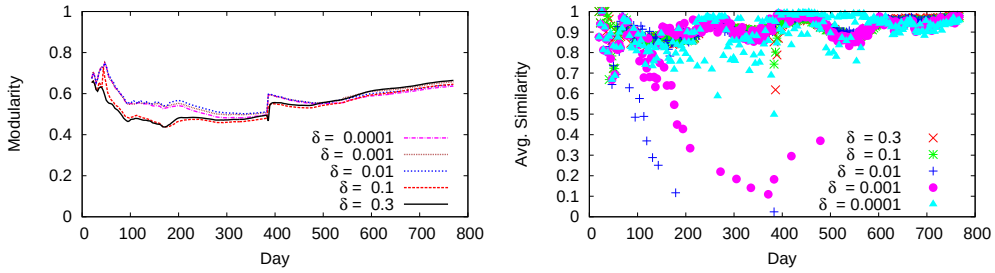
Several algorithms are designed to optimize modularity. [132] proposes a simple method to optimize modularity, reducing complexity to $O(n^3)$. [131] improves the algorithm further using hierarchical clustering method and its complexity is $O(n^2)$. [35] further reduces the complexity to $O(m \cdot d \cdot log(n))$ using balanced binary trees and max heaps. [172] improves the computation efficiency by avoiding unbalanced partitions.

Tracking Communities over Time. Tracking communities in the presence of network dynamics is a critical step in our analysis of network dynamics at

different scales. Prior work proved that dynamic community tracking is an NP-hard problem [164]. For scalability and efficiency, we use the similarity-based community tracking mechanism, which is a modified version of [65] that provides tighter community tracking across snapshots using the incremental version of the Louvain algorithm [25]. At a high level, we use incremental Louvain to detect and track communities over snapshots, and use community similarity to determine when and how communities have evolved.

Similarity-based Community Tracking. Louvain [25] is a scalable community detection algorithm that significantly improves both modularity and efficiency using greedy local modularity optimization. It uses a bottom up approach that iteratively groups nodes and communities together, and migrates nodes between communities until the improvement to modularity falls below a threshold δ. To the best of our knowledge, Louvain is the only community detection algorithm that scale to graphs with tens of millions of nodes. In this section, we use the source code for Louvain algorithm from the authors [25].

Our approach leverages the fact that Louvain can be run in incremental mode, where communities from the current snapshot are used to bootstrap the initial assignments in the next snapshot. Given how sensitive community detection is to even small changes in modularity, this approach enables more accurate tracking of communities by providing a strong explicit tie between snapshots. Finally, we follow the lead of [65], and track communities over time by computing the similarity between communities. Similarity is quantified as community overlap and is computed using set intersection via the Jaccard coefficient.



(a) Modularity

(b) Average Community Similarity

Figure 3.5: Tracking communities over time and the impact of δ.

Choosing δ**.** The δ threshold in Louvain is an important parameter that controls the trade off between quality of community detection and sensitivity to dynamics. If δ is too small, the algorithm is too sensitive, and over-optimizes to any changes in the network, needlessly disrupting the tracking of communities. If δ is too large, the process terminates before it optimizes modularity, and it produces inaccurate communities.

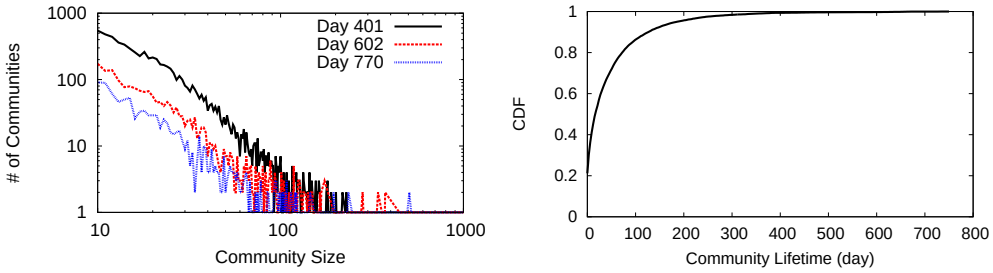
Choosing the best value for δ means optimizing for the dual metrics of high modularity and robustness (insensitivity) to slight network dynamics. First, we use network-wide modularity as a measure of modularity optimization for a given δ value. Second, to capture robustness to network dynamics, we use community similarity [65]: the ratio of common nodes in two communities to the total number of different nodes in both communities. More specifically, for two consecutive snapshots, we compute the average similarity between communities that exist in both snapshots. We run the Louvain algorithm on Renren dynamic graph snapshots generated every 3 days. We start from Day 20, when the network is large enough (64 nodes) to support communities, and only consider communities larger than 10 nodes to avoid small cliques.

We scale δ between 0.0001 and 0.3, and plot the resulting modularity and average similarity in Figure 3.5. As shown in Figure 3.5(a), in all snapshots the modularity for all thresholds is more than 0.4. According to prior work [93], modularity ≥ 0.3 indicates that Renren has significant community structure. As expected, a threshold around 0.01 is sensitive enough for Louvain to produce communities with good modularity. Note that the big jump in modularity on Day 386 is due to the network merge event. Figure 3.5(b) shows that thresholds 0.0001 and 0.001 produce lower values of average similarity (*i.e.* they are less robust and more sensitive) compared to higher thresholds between 0.1 and 0.3. Thus, Louvain with $\delta > 0.01$ generates relatively good stability of communities between snapshots.

Based on the results in Figure 3.5, we repeat the Louvain algorithm within a finer threshold range of 0.01 to 0.1. We find that a threshold value of 0.04 provides the best balance between high modularity and similarity. We use $\delta = 0.04$ to track and measure dynamic communities in the rest of our analysis on the dataset.

Community Statistics Over Time We now leverage the Louvain-based community tracking technique to analyze the dynamic properties of Renren communities.

Community Size. Our goal is to understand not only the instantaneous community size distribution, but also how the distribution changes over time as the network evolves. Thus, we compute the distributions for days 401, 602, and 770, 3 specific snapshots roughly evenly spaced out in our dataset following the network merge event. We plot the resulting community size distributions

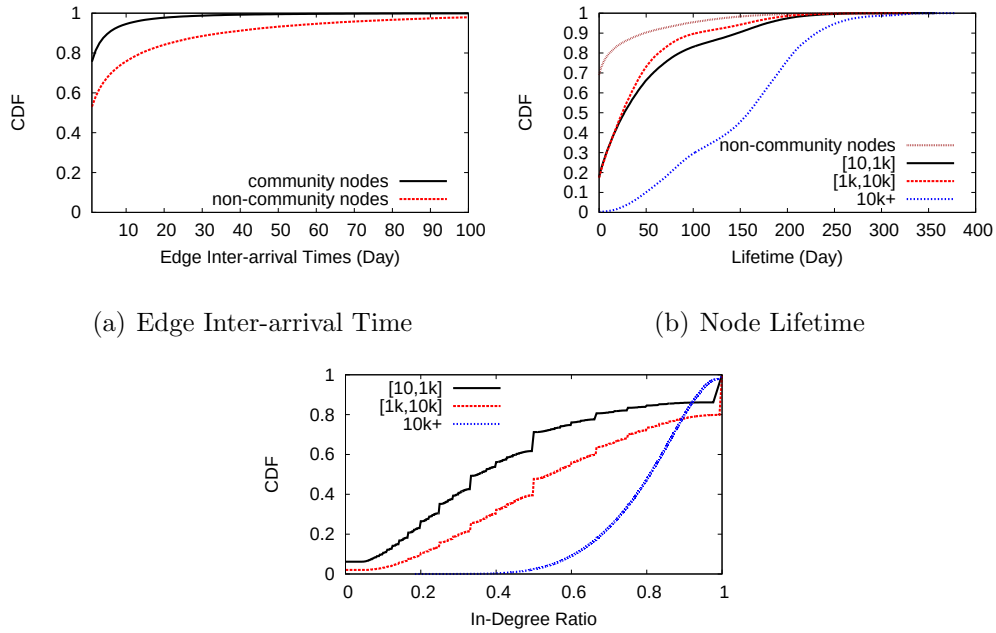


(a) Community Size Distribution (b) CDF of Community Lifetime

Figure 3.6: Analysis on the evolution of communities.

in Figure 3.6(a). The figure shows that the three snapshots consist of a large number of small communities and a long tail of large communities, consistent with the power-law distribution. This is consistent with other daily snapshots as well. More importantly, these snapshots show a gradual trend towards larger communities. Over the year of time between snapshots 401 and 770, the number of small communities shrunk by an order of magnitude. In turn, the sizes of the largest communities increase significantly.

Community Lifetime. In a dynamic network, how long a community remains in the network is another important statistical property. By using our community identification method between snapshots, we measure the distribution of community lifetime. Figure 3.6(b) shows that most of the communities only stay in the network for a very short period of time. Specifically, 20% of communities have lifetimes of less than a day, meaning that they disappear in the next snapshot after they are first detected. 60% of the communities have lifetimes less than 30 days, at which point they are merged into other communities. This shows an extremely high level of dynamics at the community level.



(a) Edge Inter-arrival Time

(b) Node Lifetime

(c) Edge In-Degree Ratio

Figure 3.7: Comparing activity of users inside and outside communities.

Impact of Community on Users. To understand how communities impact users' activity, we compare edge creation behaviors of users inside communities to those outside of any community. Overall, our results show that community users score higher on all dimensions of activity measures, confirming the positive influence of community on users.

Edge Inter-arrival Time. Figure 3.7(a) plots the CDF of edge inter-arrival times for community and non-community users. The considerable distance between the two curves confirms that community users are more enthusiastic in expanding their social connections than non-community users.

User Lifetime. Next, we examine how long users stay active after joining the network, and whether engagement in a community drives up a user's activity span. We define a user i's lifetime as the gap between the time i builds her last edge and the time i joins the network.

Figure 3.7(b) plots the CDF of user lifetime for users in different size communities as well as non-community users. $[x, y]$ represents communities of size between x and y. We find that the lifetime distribution depends heavily on the size of the community. The larger the community is, the longer its constituent user's lifetimes are. Compared to non-community users, users engaging in a community tend to stay active for a longer period of time. This confirms the positive impact of community on users.

In-Degree Ratio. We also study how users within each community connect to each other. We compute each user's in-degree ratio, *i.e.* the ratio of her edge count within her community to her degree. Figure 3.7(c) shows the CDF of the in-degree ratio for users in communities of different sizes. We observe that users in larger communities have a larger in-degree ratio, indicating that they form a greater percentage of edges within their own community. In particular, 18-30% of nodes only interact with peers in their own communities, and the portion of these nodes grows with the community size. These results show that like offline communities, online social communities also encourage users to interact "locally" with peers sharing mutual interests.

Summary. Our efforts on tracking and analyzing the evolution of communities lead to the following key findings:

- *The Renren social network displays a strong community structure, and the size of the communities follows the power-law distribution.*
- *The majority of communities are short-lived, and within a few days they quickly merge into other larger communities.*
- *The membership to a community has significant influence on users' activity. Compared to stand-alone users, community users create edges more frequently, exhibit a longer lifetime, and tend to interact more with peers in the same community.*

3.3.3 Merging of Two OSNs

On December 12, 2006 the OSN Xiaonei merged with another OSN called 5Q. This combined entity became the Renren that exists today. Our access to the graph topological and temporal data that characterizes this merge gives us a unique opportunity to study how this network-level event impacts users' activity.

In this section, we analyze the forces at work during the merge. First, we look at the edge creation activity of users over time in order to isolate users that have become inactive. This enables us to estimate how many duplicate accounts there were between Xiaonei and 5Q. Second, we examine edge creation patterns within and between the two OSNs, and show that user preferences vary by OSN and over time. Finally, we calculate the average distance between users in each group to quantify when the two distinct OSNs become a single whole.

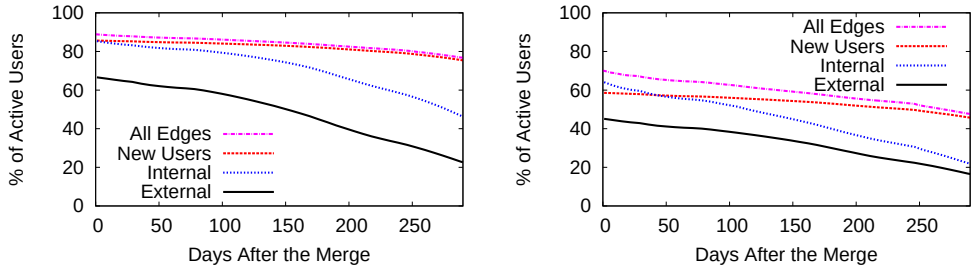
Definitions. In this section, we investigate the details of the merge between Xiaonei and 5Q. To facilitate this analysis, we classify the edges created after

the merge into three different groups. *External edges* connect Xiaonei users to 5Q users, whereas *internal edges* connect users within the same OSN. *New edges* connect a user in either OSN with a new user who joined Renren after the merge. Time based measurements are presented in "days after the merge," *e.g.* one day after the merge is day 387 in absolute terms, since the merge occurs during day 386 of our dataset.

User Activity Over Time. We start to address the question: *how many duplicate accounts were there on Xiaonei and 5Q?*, by examining the number of active Xiaonei and 5Q users over time. Users with accounts on both services were prompted to choose one account or the other on their first log-in to Renren after the merge. However, the discarded accounts were not deleted from the graph. Thus, it is likely that any accounts that are inactive on the first day after the merge are discarded, duplicate accounts.

Here we define a user as "active" if it has created an edge within the last t days. In our data, 99% of Renren users create at least one edge every 94 days (on average), hence we use that as our activity threshold t. Since our minimum activity threshold is $t = 94$ days, we cannot determine whether users have become inactive during the tail of our dataset. Thus, instead of 384 days of data after the merge, we only show 290 days in the results.

Figure 3.8(a) shows the number of active users over time for Xiaonei, while Figure 3.8(b) focuses on the 5Q users. Each "all edges" line highlights the number of users actively creating edges in each group. Both figures reveal that 11% of Xiaonei accounts and 28% of 5Q accounts are immediately inactive. Thus, it is likely that at least 39% of users had duplicate accounts on Xiaonei and 5Q



(a) Active users over time for Xiaonei

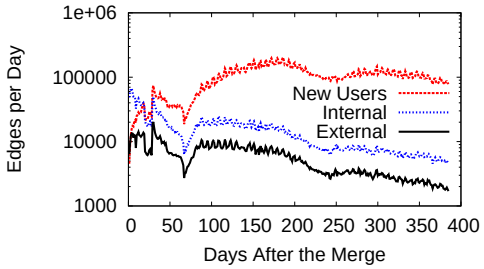
(b) Active users over time for 5Q

Figure 3.8: The number of active users over time after the network merge event.

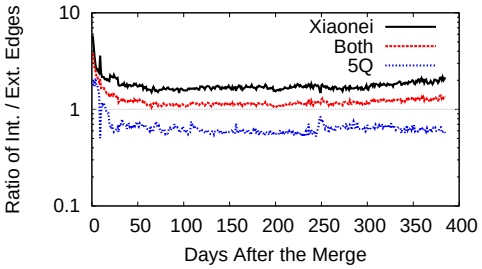
before the merge. Interestingly, users demonstrate a strong preference for keeping Xiaonei accounts over 5Q accounts.

As time goes on, the number of active accounts in each group continues to drop. Presumably, these users lose interest in Renren and stop generating new friend relationships. After 284 days, the number of inactive Xiaonei accounts doubles to 23%, while on 5Q, 52% of accounts are inactive. The relative decrease in active accounts over time (12% on Xiaonei versus 24% on 5Q) demonstrates that Xiaonei users are more committed to maintaining their OSN presence. This observation corresponds to our earlier finding that users with duplicate accounts tended to keep their Xiaonei accounts. Xiaonei users form a self-select population of more active OSN users when compared to 5Q users.

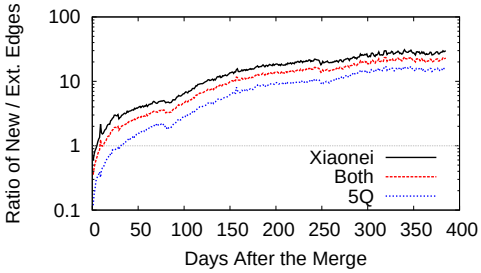
The "new users," "internal," and "external" lines give the first glimpse of the types of connections favored by Xiaonei and 5Q users. For each line, a user is considered active only if they have created an edge of the corresponding type in the last 94 days. Users in both graphs show similar preferences: edges to new users are most popular, followed by internal and then external edges. The large activity gap between internal and external edges highlights the strong homophily



(a) Active users over time for Xiaonei



(b) Ratio of the internal/external edges per day



(c) Ratio of new/external edges per day

Figure 3.9: Analysis of edge creation over time after the network merge event.

among each group of users. Internal and external edge creation activity declines more rapidly than edges to new users. This makes sense intuitively: the number of Xiaonei and 5Q users is static, and hence the pool of possible friends slowly empties over time as more edges are created.

Edge Creation Over Time. Next, we switch focus to look at the characteristics of edges, rather than individual users. By looking at the relative amounts of internal, external, and edges to new users that are created each day, we can identify what types of connections are driving the dynamic growth of Renren after the merge.

Figure 3.9(a) shows the number of internal, external, and new edges created per day. Initially, internal and external edges are more numerous than edges to new users. However, 3 days after the merge new edges begin to outnumber external edges, and by day 19 new edges out pace internal edges as well. This result demonstrates that new users quickly become the primary driver of edge creation, as opposed to new edges between older, established users. This is not surprising: since Renren is growing exponentially, the number of new users eventually dwarfs the sizes of Xiaonei and 5Q, which remain static.

We now ask the question: *are there differences between the types of edges created by Xiaonei and 5Q users?* Although Figure 3.9(a) demonstrates that internal edges always outnumber external edges, the reality of the situation is more complicated when the edges are separated by OSN.

Figure 3.9(b) plots the ratio of internal to external edges over time for Xiaonei and 5Q. Initially, users on both OSNs favor creating internal edges (*i.e.* the ratio is >1). However, by day 16, the ratio for 5Q users starts to permanently favor external edges. The reason for this strange result is that Xiaonei users create more than twice as many edges than 5Q users. In our dataset Xiaonei users create 3.9 million internal edges, while 5Q users only create 1.5 million. However, unlike internal edges, external edges affect the statistics for *both* groups. Thus, the number of external edges (2.2 million total in our dataset) is driven by the more active user base. Even though Xiaonei users create less external edges than internal edges, the number is still proportionally greater than the number of internal edges created between 5Q users. The "both" line in Figure 3.9(b) is always >1 because Xiaonei users create more edges overall, which weights the average upwards.

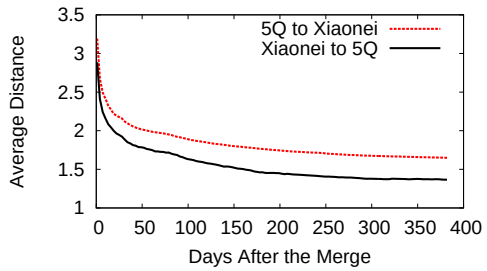


Figure 3.10: Distance between the two OSNs over time

Figure 3.9(c) plots the ratio of edges to new users versus external edges over time for Xiaonei and 5Q. This plot reveals that the inflection point where users switch from preferring external edges to new edges is different for the two OSNs. The ratio becomes ≥ 1 for Xiaonei 5 days after the merge, whereas 5Q takes 32 days. Despite these differences, both OSNs demonstrate the same overall trend for the ratio to eventually tip heavily in favor of edges to new users.

Distance Between Xiaonei and 5Q. Finally, we examine the practical consequences of edge creation between Xiaonei and 5Q. Our goal is to answer the question: *at what point do Xiaonei and 5Q become so interconnected that they can no longer be considered separate graphs?*

To answer this question, we calculate the distance, in hops, between users in each group. Intuitively, the distance between the groups should decrease over time as 1) more external edges are created, and 2) more internal edges increase the connectivity of users with external edges. In our experiments, we select 1,000 random users from each OSN on each day after the merge and calculate the shortest path from each of them to *any* user in the opposite OSN. Thus, the lowest value possible in this experiment is 1, *e.g.* the randomly selected user has

an external edge directly to a user in the opposite OSN. New users and edges to new users are not considered in these tests.

Figure 3.10 shows that the average path length between the two OSNs rapidly declines over time. Although average path lengths for both OSNs initially start above 3 hops, within 47 days average path lengths are <2. Path lengths from Xiaonei to 5Q are uniformly shorter, and by the end of the experiment the average path length is <1.5.

The distance between Xiaonei and 5Q rapidly approaches an asymptotic lower bound in Figure 3.10. Once this bound is reached, it is apparent that the graphs can become no closer together. Thus, we conclude that by day 50, when both lines begin to flatten and approach the lower bound, Xiaonei and 5Q can no longer be considered separate OSNs. These results demonstrate how quickly the two disjoint OSNs can merge into a single whole, even when edge creation is biased in favor of internal edges (see Figure 3.9(b)).

Summary. Our analysis of the network merge produces several high-level conclusions:

- *There were a large number of duplicate accounts between Xiaonei and 5Q that become inactive immediately after the merge.*
- *Edges to new nodes quickly become the driving force behind edge creation.*
- *Despite user's preference against external edges, Xiaonei and 5Q very quickly merge into a single, well connected graph.*

3.3.4 Summary of Observations

In this section, we focus on analyzing social network dynamics at different levels of scale, including dynamics at the level of individual users, dynamics involving the evolution of communities, and dynamics involving the merging of two independent online social networks.

Our analysis produced three significant findings of dynamics at different scales. First, at the individual node level, we found that the preferential attachment model gradually weakens in impact as the network grows and matures. In fact, edge creation in general becomes increasingly driven by connections between existing nodes as the network matures, even as node growth keeps pace with the growth in overall network size. Second, at the community level, we use an incremental version of the popular Louvain community detection algorithm to track communities across snapshots. We empirically analyze the impact of community on edges, and find that communities, especially large communities, have significant impact on their inside users' activities. Finally, we analyze detailed dynamics following a unique event merging two comparably-sized social networks, and observe that its impact, while significant in the short term, quickly fades with the constant arrival of new nodes to the system.

3.4 Detecting Self-Similarity in Edge Creation

To develop a model that can capture the network dynamic events in absolute time, we explore the temporal properties of the edge creation process in Renren. In recent years, as an important statistical property, *self-similarity* has been found in a variety of contexts, including local network traffic, wide-area network traffic, file

system accesses, and web traffic requests [39, 66, 98, 140, 177]. Since self-similar traffic has very different statistical properties, *e.g.* significantly higher burstiness than conventional processes, it cannot be captured by conventional traditional randomized models [140]. Influenced by the lessons from those areas, in this section we measure the existence of *self-similarity* in social network dynamics. If any exists, a complete dynamic model needs to account for it.

In this section, we describe our efforts to search for and identify the presence of self-similarity in the edge creation process of Renren. Specifically, we examine edge creation events aggregated across users over time and study the time series representing the total number of newly-generated edges per time unit. Note that our analysis efforts focus on edge creation, mainly because an exploratory analysis of the Renren data revealed no particular structure underlying the observed node creation events.

3.4.1 Background

In this section, we briefly introduce the notion of self-similarity, and then describe three popular methods used to measure self-similarity

Self-Similarity. For a continuous or discrete time process, self-similarity refers to the scale invariance behavior of the process [22, 98, 38]. Intuitively, it means that the statistic properties of the process look similar at different time scales. This type of structure is commonly associated with fractals, but has been found in a variety of contexts in computing systems and networks, including web traffic [39], file system accesses [66], and traffic in both wide area networks [140] and local Ethernet networks [98]. For self-similar traffic, the aggregation of a large number

of bursty sources produces bursty data, unlike conventional Poisson processes that tend to look uniform at large time scales.

To formally define self-similarity, let $X = \{X_i : i = 0, 1, 2, ...\}$ be a *covariance stationary* stochastic process whose autocorrelation function $r(k) \propto k^{-\beta}$ $(0 < \beta < 1)$ as $k \to \infty$. For each integer m $(m > 0)$, we form a new process $X^{(m)}$ containing the averaged values of X in disjoint blocks of size m. That is, the j^{th} element of $X^{(m)}$ is:

$$X_j^{(m)} = \frac{1}{m}(X_{(j-1)m+1} + X_{(j-1)m+2} + ... + X_{jm}). \tag{3.3}$$

If X is self-similar, then $r^{(m)}(k)$, the autocorrelation function of $X^{(m)}$, should satisfy [66, 98]:

$$r^{(m)}(k) = r(k), \text{or} \quad r^{(m)}(k) \to r(k), m \to \infty. \tag{3.4}$$

Ideally, the stochastic distributions of a self-similar process should stay invariant across all time scales. In reality, this property often exists at smaller time scales, but breaks down at large time scales due to periodic patterns and finite lifetimes [59, 66]. Thus, it is not only important to identify self-similarity, but also the range of time scales for which it is visible in the dataset [1, 59, 66].

An effective (and commonly used) metric to measure the existence of self-similarity is the *Hurst parameter* H, measureable in multiple ways [1, 98, 175]. Intuitively, H helps to capture the "burstiness" of a covariance stationary process, where a higher H corresponds to aggregate traffic with stronger bursts. Formally, $H = 1 - \beta/2$, where β is defined by the process X's autocorrelation function $r(k) \propto k^{-\beta}$. A process exhibits self-similarity if H falls in the range of $(0.5, 1)$.

Estimating H. In this section, we consider three popular methods to estimate the Hurst parameter H: *variance analysis*, *R/S analysis* and *wavelet-based analysis*.

Variance analysis [98, 140] estimates H directly from β. From Eq. (3.4), a self-similar process X satisfies

$$\log(Var(X^{(m)})) \propto -\beta \log(m), \quad m \to \infty$$

where $X^{(m)}$ are the aggregated processes introduced earlier where m is the block size and $\beta = 2(1-H)$. Thus, by linearly fitting the plot of $\log(Var(X^{(m)}))$ versus $\log(m)$, this method can estimate β and then H.

R/S analysis computes H by measuring how apparent the variability of a time series changes with the length of the time-period being considered. This can be formally captured by the R/S statistic [66, 98]. To compute H, this method divides the process X into blocks of size n, and computes the corresponding R/S statistic $R(n)/S(n)$. Because $E[R(n)/S(n)] \propto n^H$ [66] for self-similar processes, one can estimate H using the slope of $\log(E[R(n)/S(n)])$ versus $\log(n)$.

Wavelet-based analysis represents a process X by a sequence of subspaces $\{W_j\}_{j \in Z}$ where W_j is at a finer scale than W_{j-1} ($W_j \subset W_{j-1}$). This way, the method can reveal detailed properties of X at different time scales. If X exhibits self-similar scaling, its projection on the W_j subspace, Γ_j, satisfies: $E[\Gamma_j] \sim |2^{-j} v_0|^{1-2H}$. Here $2^{-j} v_0$ represents the reference frequency of the j^{th} subspace W_j while v_0 is the reference frequency of the root subspace W_0. Then H can be estimated by plotting $E[\Gamma_j]$ vs. scale j on log-log scale and applying linear regression.

Discussion. Because of the simplicity, variance analysis and R/S analysis are widely used in self-similarity studies [59, 66, 98]. However, as "eyeballing" approaches, they produce results with higher estimation errors [83, 166], which is also shown in Section 3.4.2. In contrast, wavelet-based analysis offers a principled and rigorous analysis of a given dataset's scaling property, and provide more reliable results with confidence interval in detecting self-similarity [1], which will be explored in Section 3.4.3.

3.4.2 Preliminary Analysis on Sampled Data

Our goal is to investigate if Renren's network evolution displays properties consistent with self-similarity, and if so, over what range of time scales. A key challenge we face is identifying and isolating the impact of non-stationary patterns in the edge creation data. As a first step, we limit the impact of new node arrivals on edge creation, by focusing our analysis on edges created between members of a fixed user population.

In the following, we start by briefly describing how we sample the original dataset by removing certain node arrival and other obvious non-stationary events. We then show our initial analytical findings and resulting key insights.

Data Sampling. We begin our analysis with a conservatively sampled subset of our data to remove obvious non-stationary factors that may impede any direct analysis of self-similar scaling properties. Specifically, we limit our sample to include only existing users as of December 1, 2007, and study all edge creation events between them during December 1-31, 2007, *i.e.* days 741 to 771. This sampling eliminates three factors. First, by studying only edges created between

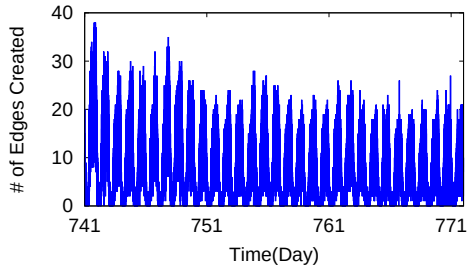


Figure 3.11: The number of new edges created per second in the sampled dataset.

members of a fixed user population, we minimize the impact of new node arrivals. Second, this month avoids the abnormal expansion of new edges around day 386 as a result of the one-time merge of two social networks (Renren and 5Q). Finally, this time period is sufficiently late in the history of Renren that it avoids the initial exponential network growth experienced by the Renren network. This data sample represents a stable growth period in Renren, and contains $18,714,712$ edges created between $6,219,531$ existing users.

Measurement Results. We now present the results using the following three heuristics: *visualization of raw data*, *variance analysis* and *R/S analysis*.

A Long-term Diurnal Pattern. Figure 3.11 visualizes the edge creation process by plotting the number of new edges created in each second over the one month (Day 741-771). It shows a clear diurnal pattern in the edge creation process. This obvious non-stationary behavior precludes any direct analysis of self-similarity.

We also confirm this from the results of the variance and R/S analysis. Figure 3.12 plots the values of $\log(Var(X^{(m)}))$ against $\log(m)$. The curve maintains a linear shape until m reaches about 10^4 seconds ($\approx$ 3 hours), and then its slope

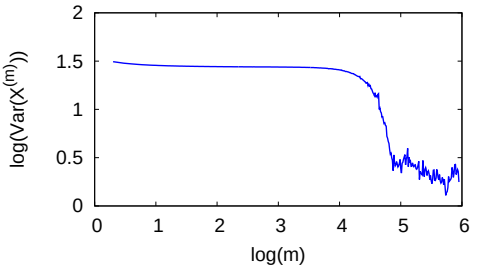


Figure 3.12: Variance analysis in the sampled dataset.

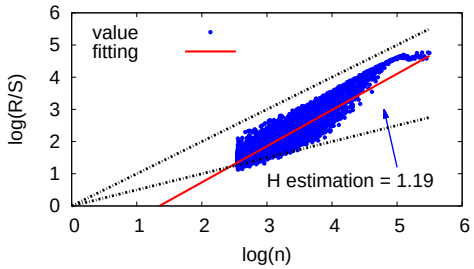


Figure 3.13: R/S analysis in the sampled dataset.

changes significantly. Similarly, Figure 3.13 plots in log-log scale individual R/S statistics as a function of the block size n (in seconds). The red straight line shows the best linear fit and its slope results in an H-estimate of $H = 1.19$, clearly outside the allowed range of $(0.5 < H < 1)$.

We also note that the appearance of such a pronounced diurnal pattern has a direct impact on subsequent efforts to model our dataset and suggests that models should include a component that accounts for this expected user-generated periodic behavior.

Self-similar Fluctuations. An interesting observation from Figure 3.11 is that the fluctuations on top of the diurnal component appear to display consistently bursty behavior. Similarly, Figures 3.12 and 3.13 both show that the measurement data only starts to lose its (straight line) shape when m or n exceeds 10^4 seconds (about 3 hours). These observations suggest that over time scales that are not significantly impacted by the presence of the observed diurnal patterns (i.e., a few hours and below), the time series of new edge creations may be consistent with self-similar scaling behavior.

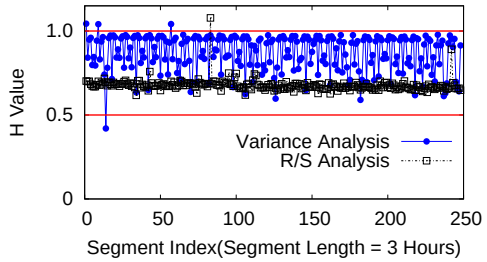


Figure 3.14: Estimates of H for 248 disjoint 3-hour segments.

We confirm this intuition by performing variance and R/S analysis on each 3-hour log segment and computing its corresponding H value. Figure 3.14 plots the results over the entire month as 248 disjoint 3-hour segments. H estimates based on the variance analysis method vary across segments, with a mean of 0.8867 and variance of 0.0108. Regarding the R/S analysis method, the obtained H-estimates remain stable across all segments, with a mean 0f 0.6752 and variance of 0.0006. For both methods, the overwhelming majority of segments (98.4% for variance, 99.5% for R/S) estimate H within $(0.5 < H < 1)$. These results provide strong evidence that the Renren edge creation process exhibits self-similarity over time scales ranging from seconds to hours.

The Reliability of our H Estimates. In the process of our preliminary data analysis, we encountered potential issues regarding the reliability of the H-estimates obtained by both the variance and R/S analysis methods. For a number of segments, the methods produced poorly-fitting linear regression lines, which in turn resulted in highly questionable estimates of H. Figure 3.15 shows an example of such a "problematic" segment (6-9am, December 6, 2007), where the quality of the line fitting is poor via variance analysis. We also plot as an inset in the figure

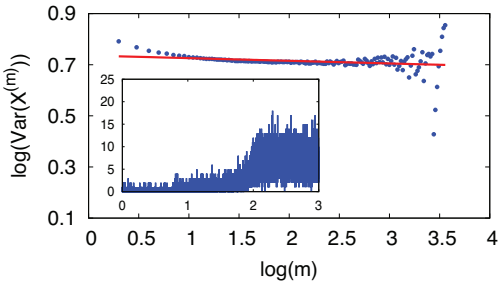


Figure 3.15: An example of poor line fitting in variance analysis.

the raw edge growth during the time period, which shows a clear non-stationary event.

To quantify the impact of such poor data fitting on the obtained H estimates, we compute for each segment the coefficient of determination R^2, which measures how well the observed data points (from each method) are represented by a straight line. Like [66], we use the criterion of $R^2 > 0.9$ to indicate that the fitting is sufficiently good to provide a reliable H estimate. Out of all segments, 38.31% of the segments have unreliable H estimates using R/S analysis vs. 70.97% using variance analysis! We note that similar reliability issues have also been reported by prior studies [83, 166].

Summary of Observations. Our initial analysis led to three main findings.

- *The edge creation process in Renren displays a typical diurnal pattern in user activity that makes the process inherently non-stationary and thus prevents a direct analysis of self-similarity.*

- *Local fluctuations on top of the periodic component display behavior consistent with self-similar scaling in 3-hour segments using variance analysis and R/S analysis.*

- *We find that both the variance and R/S analysis methods cannot provide reliable H.*

3.4.3 Wavelet-Based Analysis

To avoid most of the encountered problems in Section 3.4.2, we apply a rigorous wavelet-based method to systematically study potential self-similar scaling behavior exhibited by our dataset. In this section, we start our analysis by confirming and substantiating our preliminary results that show properties consist with self-similar properties in sampled data. Then, we remove the restriction and extend our analysis to all edge creation events. In our analysis, we refer to a segment as "abnormal" if its corresponding H estimate (including its 95% confidence interval) does not completely fall within the required range $(0.5, 1)$.

Experiment Setup. We estimate H using the wavelet software developed for self-similarity analysis [169]. By carefully choosing the number of vanishing moments N that controls v_0, the tool can systematically detect and then remove the impact of various types of deterministic trends in the dataset. Furthermore, it also relies on known theoretical properties of the resulting H-estimate to provide confidence intervals for H. In the analysis of our dataset, we choose the value of N that produces both a good fit and the smallest confidence interval.

Confirming Preliminary Results. First, we use a wavelet-based analysis to measure whether Renren's edge creation process exhibit self-similar scaling behavior in 3-hour segments as shown in Section 3.4.2. Figure 3.16 shows the H estimates with their 95% confidence intervals for all 248 disjoint 3-hour long seg-

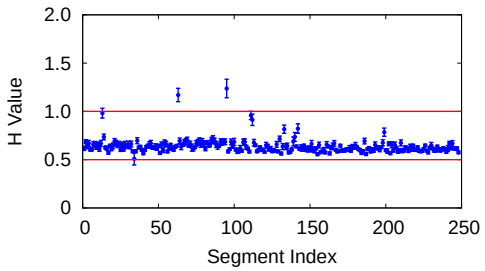


Figure 3.16: Wavelet analysis of sampled data using 3-hour segment length.

Start Time Shift	Normal Segments		Abnormal Segment Portion
	H mean	H variance	
0 hour	0.6312	0.0020	2.02%
1 hour	0.6326	0.0021	2.43%
2 hours	0.6291	0.0019	2.02%

Table 3.1: Statistics of 3-hour segments with start time shifts.

ments derived from the sampled used in Section 3.4.2. We see that only 5 segments are abnormal, and all other segments have highly consistent H estimates that are tightly clustered around $H = 0.63$. In other words, the large majority (98%) of data segments consistently produce H estimates that are well within the interval $(0.5, 1)$.

To examine the robustness of our results, we check different segment compositions. For instance, we shift the start times of each segment by 0, 1, and 2 hours, and summarize the results in Table 3.1. We notice that the mean and variance of the resulting H-estimates for all normal segments remain stable. In addition, the portion of segments deemed abnormal also remains stable at $2.02\% \sim 2.43\%$. These results provide further evidence that Renren's edge creation process is consistent with self-similar scaling over time scales in 3-hour time scale.

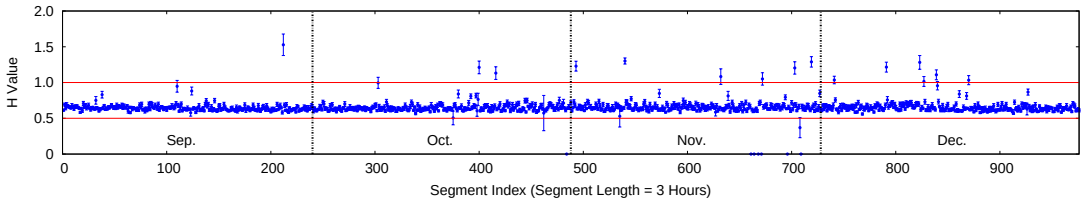


Figure 3.17: The estimate of H of all the disjoint 3-hour segments between September - December 2007 on the dataset without sampling.

Analysis Without Sampling. Next, we expand our analysis to consider the full, unsampled dataset. Our goal is to understand whether the observed self-similar scaling behavior on the sampled data is still present after including new nodes with rapid (and non-stationary) edge growth.

We examine all edge events in the year of 2007, when the network growth get stabilized after the network merge event. Similar to the results on the sampled dataset in Figure 3.16, 97% of the 3-hour segments fall into the self-similar range, with mean $H = 0.64$. Figure 3.17 shows H estimates for September-December 2007 (due to the space limit), which are representative of all other months. The result suggest with high confidence, that the same self-similar scaling property exists consistently in the edge creation process of the Renren network throughout time. These results also confirm the high reliability of the wavelet method in self-similar detection.

Summary. We apply the more reliable and accurate wavelet method to detect self-similarity in 3-hour segment length. There are two key findings.

- *The outcomes confirm prior R/S and variance results, with high confidence, that the property consistent with self-similar scaling lasts to several hours.*

- *We also note that this property holds for our full, unsampled dataset (minus the network merge event).*

3.4.4 Summary of Observations

In this section, in addition to the "eyeballing" approach, we measure the existence of self-similarity in edge creation process with three popular methods, *i.e.* variance analysis, R/S analysis and wavelet-based analysis. There are two important take-aways from our observations.

First, in long-term time scale, such as days or weeks, there is an obvious diurnal pattern in edge creation, indicating the Renren network is non-stationary. This makes the analysis of self-similarity challenging.

Second, with a more reliable wavelet-based method, we identify that local fluctuations on top of the diurnal component of the Renren data exhibit self-similar scaling properties in short-term time scale, *i.e.* 3 hours.

3.5 A model of network dynamics

Motivated by our analysis on the dynamics of Renren network structure and the self-similar analysis of Renren' edge creation process, we seek to build a complete model of social network edge dynamics. The model includes two components: a *temporal* component that produces a sequence of time-stamped events defining *when* and *how many* new edges are formed, and a *spatial* component defining *where* in the graph these new edge creations take place. Ideally, the model should produce synthetic dynamic graphs that display diurnal and self-similar properties in edge dynamics, as well as graph structural changes observed in Section 3.2 and

Section 3.3, such as graph densification, path shrinkage, local declustering and the fact that old nodes drive edge creation as the network grows. Next, we explain the model in detail, and provide validation at the end of this section.

3.5.1 The Temporal Component

Our analysis in Section 3.4 shows that for the Renren social network, the edge creation process displays a combination of (long-term) diurnal patterns and self-similar behavior over shorter time scales (3 hours). This motivates us to build the temporal component by combining two sub-modules: a *non-stationary* diurnal module that dominates at large time scales and captures the predictable cycles in user daily activities, and a *self-similar* module that produces the inherent short-term burstiness in user edge creations.

The Self-Similar Module. Prior work has demonstrated two effective methods for producing self-similar traffic. The first aggregates many ON/OFF processes and the superposition displays a self-similar traffic pattern [178]. It requires statistical knowledge of the ON and OFF periods, both following some heavy-tailed distributions. The second type constructs an $M|G|\infty$ queuing model [38, 176]. Each source arrives according to a Poisson process, and its active time follows a heavy-tailed distribution, *e.g.* the Pareto distribution. With a constant rate during each node's active time, the resulting count process $\{N_t, t = 0, 1, 2, ...\}$, where N_t is the number of active sources at time t, is self-similar. In other words, by multiplexing sources with Poisson arrivals and heavy-tailed active times, one can produce a self-similar process.

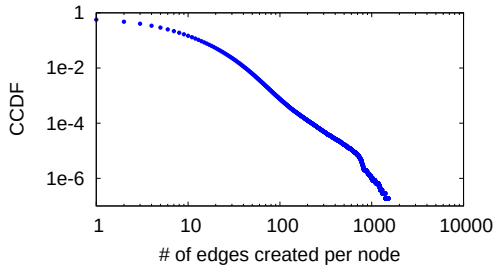


Figure 3.18: CCDF of edge # created per user in Dec. 2007.

A deeper look at our measurement data shows that the $M|G|\infty$-based method provides a more natural way and a better fit for modeling the Renren social network. This is because over time, the number of edges created per Renren user follows a heavy-tailed distribution. For example, Figure 3.18 plots the distribution of the number of edges created per user during December 2007, which clearly displays a heavy-tailed pattern. Assuming each user creates edges at a constant rate, the active time of a user is directly proportional to its count of edges created. This implies that each Renren user's active time also follows a heavy-tailed distribution, consistent with the construction of the $M|G|\infty$-based method.

We build the self-similar module similar to a standard $M|G|\infty$ process [38]. Each user starts its active time period following a Poisson process with rate λ. As it starts, we determine its active time duration T_i (seconds) based on a Pareto distribution $P(X > x) = (\frac{x_m}{x})^\alpha$, $(x \geq x_m,\ 1 < \alpha < 2)$. Assuming that each user creates edges at a constant rate γ/s, we can calculate the total number of edges created by user i by $T_i \cdot \gamma$. Since an edge is created by two users, we derive the number of edges S_t created at time t from the number of active users N_t: $S_t = \gamma \cdot N_t/2$, and get the self-similar module $\{S_t, t = 0s, 1s, 2s, ...\}$.

The Diurnal Module. We extract the non-stationary diurnal component by subtracting the self-similar component from the original edge creation process. Suppose the number of original edge creation in Renren is O_t at time t. Then the subtraction produces a process $\{U_t = O_t - S_t, t = 0s, 1s, 2s, ...\}$. We then apply a sliding window over U_t to derive a smooth long-term curve, and then fit this curve with a periodic function, *i.e.* Sine, to produce D_t, the diurnal module for our temporal component.

Integrating the Two Modules. We combine S_t and D_t together as our targeted edge creation process E_t: $\{E_t = S_t + D_t, t = 0s, 1s, 2s, ...\}$. Since the diurnal component D_t may generate negative values, we set a minimum for the sum to be 0.

Note that we design this temporal component to characterize new edge events aggregated across all the users, capturing the key properties of edge dynamics that are consistent with self-similar scaling and diurnal patterns. Thus this component only generates timestamps of new edges (in terms of the total number of edges created in each second), but do not associate any of these new edges to specific users. The actual distribution of edge events across users is performed by the spatial component, which we will describe next.

3.5.2 The Spatial Component

To determine where each new edge is created in the network evolution process, we first highlight two key observations made during our analysis of dynamics in the Renren network structure, *i.e.* Section 3.2 and 3.3. *First*, in Section 3.3, we observed that past an initial rapid growth phase, new edge creation was dominated

by pairs of existing nodes (>80%). This result differs from existing graph models, which generally assume that new node arrivals consistently drive edge creation regardless of network size. *Second*, the results in Section 3.2 show that the Renren network displays three structural properties over time: graph densification, distance shrinkage, and high but decreasing clustering coefficient (CC). Existing graph models [104, 118, 29, 6, 5] capture only a subset of these properties.

Intuition. We consider a stable social network in a state of ongoing growth. The results in Section 3.2 show that past a fast initial period of explosive growth (from 0 to 0.1M users in Renren), the arrival rate of new users becomes relatively small compared to existing active users. At this point, continuous friend discovery between existing users dwarfs the initial bursts of edge creations triggered by new user arrivals. Therefore, in our model, we use inter-arrival gaps between new users as iterations to drive the formation of new edges between existing users.

Our intuition is to focus on the creation of edges between existing users following the arrival of each new user. For simplicity, we assume a new user u_i creates an edge before the arrival of the next user u_{i+1}, and after this edge creation u_i immediately becomes an "existing user." We hypothesize that existing users are often introduced to groups of friends, either discovering the presence of an offline friend (and other mutual friends), or creating new groups of friends via common interests or social applications. To capture this intuition, in each iteration, our model selects two existing nodes u and v at random, and connects node u repeatedly to multiple users in node v's neighborhood. Here node v can be an existing friend of u or a previously unknown "stranger." The continuous formation of random connections between existing users shrinks average path lengths and lowers

the clustering coefficient by building shortcuts between nodes, while connecting friends of friends slows the rate of declustering.

A Detailed Spatial Component. The spatial component is strongly dependent on the temporal component to determine the maximum number of edges created in any iteration (*i.e.* between two node arrivals). Let $F(n)$ represent the number of edges in the network when the network contains n nodes. Then $F(i+1) - F(i)$ represents the total number of edges created between the arrivals of u_i and u_{i+1}, which can be computed by the results generated by the temporal model. With the knowledge of node arrival time statistics, *i.e.* t_i and t_{i+1}, we can estimate the total number of edges k to create between t_i and t_{i+1} as $k = F(i+1) - F(i) = \sum_{t=t_i}^{t_{i+1}} E_t$.

Specifically, our proposed edge formation process is defined as follows. We drive the process using a parameter p, which defines the probability a node is selected in the recursive edge creation process between existing nodes.

1. When a new node u_i joins the network, $k = F(i+1) - F(i)$.
2. **Edge creation by the new node:** The new node u_i randomly selects an existing node u_j to connect. $k = k - 1$. Now u_i becomes an existing node.
3. **Edge creation between existing nodes:** Randomly select two existing nodes u and v. If they are not connected, connect them and set $k = k - 1$. Then node u starts the following steps (a)-(c) to connect neighbors of node v and repeat them until all the required edges have been created (*i.e.* $k = 0$) or there are no more nodes to connect. Each time an edge is created, set $k = k - 1$.

(a) Generate a random number x following the geometric distribution with mean $(1-p)^{-1}$.

(b) Randomly selected the neighbors of node v that do not connect node u until reaching any of the three situations: 1) x neighbors are selected; 2) there is no more edges that need to be created, *i.e.* k=0; 3) all available neighbors of node v are selected. Let R=$\{r_1, r_2, \ldots, r_x\}$ be the set of selected nodes.

(c) For each node $r_i \in R$, node u connects node r_i and repeats steps (a)-(b) on node r_i.

4. If more edges need to be created ($k \neq 0$), repeat step 3.

Discussion. The existing model most similar to our new model is the Forest Fire model [104], which simulates network growth by creating edges between each new node to a set of existing nodes. A new node joining the network randomly connects to an existing node and some of its neighbors; this repeats across the network, like a fire burning through a forest. This "burning process," and the recursive edge creation process between existing nodes in our model both act to produce high clustering coefficients, by recursively connecting to neighbors of neighbors.

Three key differences separate our model from Forest Fire. *First*, our model captures the observation that existing nodes drive edge creation in a stable growth network. *Second*, our model produces decreasing clustering coefficients by connecting pairs of random existing nodes. Forest Fire does not capture this property because it always forms close triangles in each node's neighborhood, leading

to relatively high clustering coefficients unlikely to decrease over time. *Third*, our model can be accurately calibrated to the observed dynamics of an existing network trace, by incorporating the network growth function from the temporal model. This additional flexibility makes it more attractive for generating realistic dynamic network traces.

3.5.3 Model Validation

Having described the model for network edge dynamics, we now validate this model using the Renren dataset. Specifically, we calibrate the model using the Renren measurement, and then use the calibrated model to generate a set of synthetic dynamic graphs. We then compare these graphs to the original data in terms of both temporal and spatial properties. Because the output of the temporal component is used as an input to the spatial component, the corresponding validation on the spatial component also validates the complete model with both components.

Validating the Temporal Component. To demonstrate the accuracy of the temporal component, we calibrate the component with the Renren dataset of the month of December 2007, the same dataset used in our self-similarity analysis.

Calibrating the Self-Similar Module. We construct the self-similar process according to the $M|G|\infty$ model with Poisson arrival rate λ, whose active time period follows a Pareto distribution with parameters α and x_m. Consider the Renren edge creation data collected in December 2007 where $7,246,621$ nodes have created edges. We can estimate the corresponding value of λ in this period by the

	R/S	Variance	Wavelet
H Estimation	0.6784	0.6322	0.6935
Confidence Interval	–	–	0.0099

Table 3.2: Self-similar analysis on the self-similar component of the synthetic data.

average active node count in the unit of seconds, *i.e.* $\lambda \approx 2.7$/s. To derive the active time (in seconds) statistics, we leverage a proven relationship between H and α [38, 98]: $H = (3 - \alpha)/2$. Because our measured average H value for the December 2007 dataset is 0.65, we have $\alpha = 1.7$. Finally, assuming a node creates edges at a constant rate of 1/s, the average number of edges created per node is then equal to the average active time across all the nodes, *i.e.* the mean of the Pareto distribution $x_m * \alpha/(\alpha - 1)$. By measuring the average edges created per node in December 2007, we get $x_m \approx 3.2$.

Using the $M|G|\infty$-based method with $\lambda = 2.7$/s, $\alpha = 1.7$ and $x_m = 3.2$, we generate a synthetic trace that represents the edge creation process contributed by the self-similar module. To validate that the resulting trace is indeed consistent with the designed-for self-similar scaling behavior (i.e., $H = 0.65$), we apply the earlier-described R/S, variance and wavelet analysis methods. Table 3.2 summarizes the results of this exercise and shows that the estimated H values are 0.68, 0.63 and 0.69, respectively (95% confidence interval is 0.0099).

Calibrating the Diurnal Module. We calibrate the diurnal module by first subtracting the synthetic trace generated by the self-similar module from the original edge creation data. We apply a sliding window of size 1 hour and a step size of 1 second to smooth the subtraction result over time. The smoothed data for December 2007 is well-fitted by the sine function: $9.7 \sin(7.27 \cdot 10^{-5}\, t + 3.56) - 0.003$.

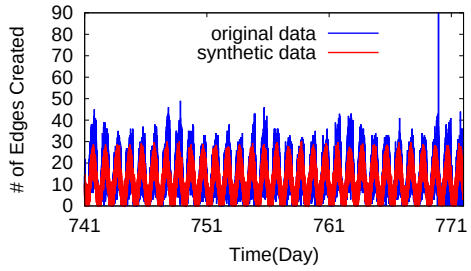


Figure 3.19: Synthetic edge creation process in Dec. 2007

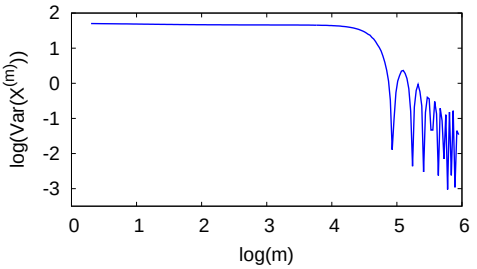


Figure 3.20: Variance analysis of synthetic edge creation.

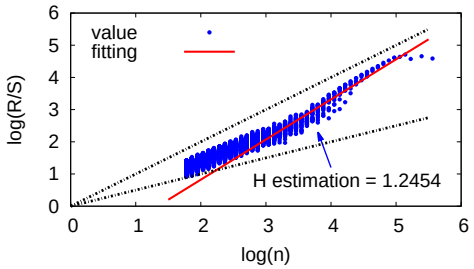


Figure 3.21: R/S analysis of synthetic edge creation process.

Validation Results. We sum the synthetic traces produced by the above two sub-modules to build a single synthetic edge creation trace, and then compare it to the original data. Repeating the process *5 times* produces extremely consistent outcomes. The total edge counts are very similar, with the average ratio between synthetic traces and the original of 1.007 with variance $< 10^{-6}$. Figure 3.19 plots a sample of one synthetic trace together with the original trace (for December 2007) where the synthetic data displays diurnal patterns similar to the original data.

We further compare the two traces using the self-similarity analysis over both longer time scales and short time scales discussed in Section 3.4. Figures 3.21 (R/S analysis) and 3.20 (variance analysis) demonstrate that the synthetic trace

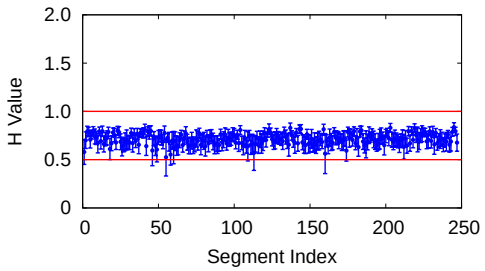


Figure 3.22: Wavelet analysis of synthetic edge creation process in segments of 3 hours.

exhibits the very same issues that plagued our preliminary analysis of the original data; e.g., scaling behavior changes drastically for time scales larger than a few hours, H-estimates are outside the theoretical range $(0.5, 1.0)$, and non-stationary diurnal patterns preventing a direct scaling analysis of the data.

Next we apply the wavelet-based analysis method to examine the self-similar nature of the synthetic trace over 3-hour segments. Figure 3.22 plots the resulting H-estimates for each segment together with their 95% confidence intervals. We see that the H-estimates for the synthetic trace also fall consistently between 0.5 and 1 with an exception of 4.03%, which closely matches the 3% exception seen from the original data. The average H value for the synthetic trace is around 0.75, again very similar to that of the original trace (mean H=0.65) as shown in Figure 3.16.

Together, these results demonstrate that our temporal component can accurately capture the diurnal patterns and properties consistent with self-similarity in short time scales displayed by the original Renren data. The contributions of the two sub-modules also explain why self-similarity exists in short time scales but breaks down in longer ones.

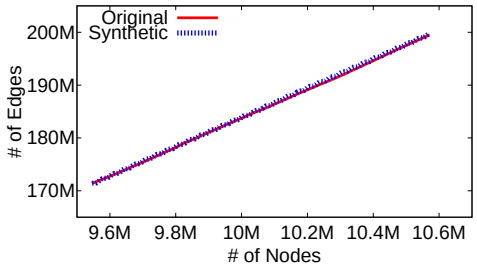


Figure 3.23: The synthetic network growth trace of Dec. 2007 generated by the temporal component vs. the original network growth trace in Dec 2007.

Connecting the Temporal and Spatial Components. Recall that the spatial component of our model uses the temporal component to compute the number of edges created between each pair of node arrivals, *e.g.* between arrival of i^{th} and $i+1^{th}$ nodes: $\sum_{t=t_i}^{t_{i+1}} E_t$. This requires us to accurately estimate t_i, the arrival time of each node. From our analysis, we find that no properties consistent with self-similarity exist in the node arrival process. Instead, we find that a Poisson process with arrival rate λ_{new} can accurately model new node arrivals, where λ_{new} is estimated as the average number of new nodes arriving per second. This is consistent with the observation in [140], that while packet arrivals appear better modeled using self-similar processes, Poisson processes can effectively capture user sessions. With the new node arrival process modeled by the Poisson process, the network edge growth $F(i)$ can be predicted as a function of the network node count i, where $F(i+1) - F(i) = \sum_{t=t_i}^{t_{i+1}} E_t$. Figure 3.23 shows that our solution $F(i)$ can accurately predict the network growth in Dec. 2007.

Validating the Spatial Component Because of extremely costly calibration process and the network merge event on Dec. 12, 2006, it is impractical to calibrate the model using the entire Renren dataset. Instead, we use two data segments to

Graph	# of Nodes	# of Edges	Avg. Deg	Avg. path	Avg. CC
2006 Original	624,364	8,258,266	26.45	4.16	0.159
2006 Synthetic	624,364	8,721,927	27.93	4.46	0.183
2007 Original	1,751,146	18,203,520	20.79	4.87	0.156
2007 Synthetic	1,751,146	18,305,972	20.9	4.78	0.159

Table 3.3: Statistics of the original graph and the synthetic graph generated by our spatial component.

validate the spatial component. The first segment (referred to as *2006 Original*) covers the launch of the network (Nov. 21, 2005) to Dec. 11 2006, right before the merge event. The last snapshot of the graph includes 624K nodes and 8M edges. This represents the "early" period of the Renren network. The second segment (*2007 Original*) covers the first two months of 2007, and its last snapshot has 1.75M nodes and 18M edges. This represents the "stable growth" period of the Renren network. We use the snapshot on Dec. 31, 2006 as the initial graph and apply our spatial component to model the network evolution. Table 3.3 summarizes the observed network statistics for the two segments.

Calibrating Spatial Component. For each of the two segments, we calibrate the two parameters of the spatial component: *network growth function* $F(n)$ and *node selection probability* p. For the 2007 segment, we derive $F(n)$ from the temporal component. For the 2006 segment, since the 2006 network is not stable and large enough to display significant temporal patterns, we manually fit the network growth by a polynomial function. Figure 3.24 shows the $F(n)$ estimation results for both segments, which closely match the original data. Next, we determine p by generating synthetic graphs with p varying between 0.1 and 0.9, and choose the optimal p value that produces graphs with network distance and clustering coefficient most similar to the original data. The resulting p values

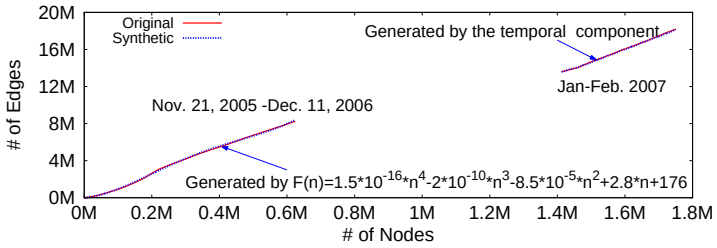


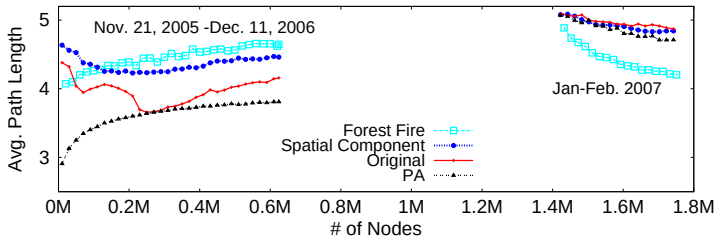
Figure 3.24: Network growth fitted by $F(n)$.

are different for the two segments: 0.7 for the 2006 segment and 0.5 for the 2007 segment.

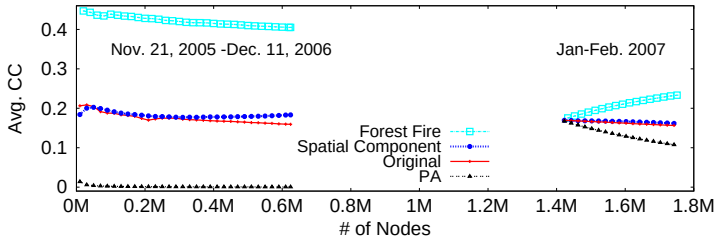
Validation Results. Using the calibrated component, we generate synthetic dynamic graphs for the two data segments. As shown in Table 3.3, the synthetic graphs statistically match the original graphs in terms of average degree, average path length and average clustering coefficient. The goal of our validation is to understand whether the synthetic graphs display graph densification, average path length shrinkage and decreasing clustering coefficient (CC). Figure 3.24 confirms that the synthetic graphs can accurately capture the densification property. Thus in the following, we focus on evaluating the dynamic properties of average path length and average clustering coefficient in the synthetic graphs.

As a reference, we also include the results using the preferential attachment (PA) model, the most popular static graph model, and the original Forest Fire model, which is the most similar to our model [2]. We repeated our experiments *five times* for all three models. Due to the consistency of the results with variance ≥ 3

[2]Following a similar procedure described by [149], we modify the forest fire model to produce undirected graphs by creating undirected edges and allowing the "burning" process to proceed in both directions of an edge. To calibrate the model, *i.e.* determining the burning probability p, we sample values between $(0, 1)$ to find the best fit p where the corresponding synthetic graphs match the original graph the most in terms of network distance and clustering coefficient



(a) Avg. path length



(b) Avg. clustering coefficient

Figure 3.25: Synthetic graph dynamic properties

order of magnitude smaller than the average value, we only show the results for a single run in the following, where "Original" stands for the Renren graph, "Spatial Component" stands for the graph generated by our spatial component, "PA" means the graph generated by the preferential attachment model and "Forest Fire" is the graph generated by the original Forest Fire model.

Average Path Length Evolution: Figure 3.25(a) plots the average path length over time using our spatial component, the PA model, the original Forest Fire model and the original data, for the two time segments. For the 2006 segment, our proposed spatial component displays the most similar pattern to the original data, where the path length decreases first and then increases slightly, while the PA and the Forest Fire models lead to a different pattern where the path length continues to increase over time. For the 2007 segment, all four graphs display

a similar pattern where the average path length decreases over time. Even in this case, our spatial component is the most close to the original graph. In this segment, the behaviors of the PA and Forest Fire models change because the initial graph used to generate the synthetic graphs is a snapshot of the original data captured on Dec. 31, 2006. This helps to remove the long-term impact of preferential attachment that produces increasing average path length over time.

Average Clustering Coefficient Evolution: Figure 3.25(b) plots the results of average clustering coefficient from the three models and the original data. For the 2006 segment, both the original data and our spatial component produce an average clustering coefficient between 0.15 and 0.25, which decreases slightly over time. The clustering coefficient of the PA model stays closely to 0 while that of the Forest Fire model remains around 0.4. For the 2007 segment, again our spatial component produces nearly identical value of the original data, while the result of the PA model deviates largely and that of the Forest Fire model displays an opposite pattern. Together, these results confirm three key findings. *First*, our spatial component can accurately capture the significant local connectivity and the slowly decreasing clustering coefficient. *Second*, the PA model is unable to maintain high clustering coefficient over time, even when growing from a highly clustered graph. Finally, as indicated by our earlier analysis, the Forest Fire model produces relatively high clustering coefficients, and thus is unable to capture the key properties of Renren such as decreasing clustering coefficient.

Our validation confirms that the spatial component can accurately capture the dynamic features observed from Renren. Since our 2007 dataset takes input from the temporal component, the spatial component validation also validates the complete model with both the temporal and spatial components.

3.6 Related Work

Dynamic OSN Measurement Several studies have measured basic dynamic properties of graphs. [104] analyzed four citation and patent graphs, and proposed the forest fire model to explain the observed graph densification and shrinking diameter. [101] studied details of dynamics in four OSNs to confirm preferential attachment and triangle closure features. Similar conclusions were reached by studies on Flickr [124] and a social network aggregator [58]. [82] measured network temporal radius and found out that there is a gelling point to distribution. In addition, [6] measured weighted dynamic graphs, [3] analyzed the growth of a Korean OSN, and [170] considered temporal user interactions as graph edges instead of static friendship. Finally, [68, 90] analyzed blogspace dynamics.

Some studies focused on analyzing social network dynamics through explicitly defined groups [14, 190, 79] or disconnected components [91, 118, 81]. [90] tried to identify blog communities and detect bursts in different temporal snapshots. [136] utilized the clique percolation method [44] to identify overlapping community dynamics in mobile and citation graphs. Unlike these studies, our work focuses on the evolution of implicit communities in a densely connected, large-scale social graph.

Dynamic Community Detection and Tracking Algorithms There are two approaches to detecting and tracking dynamic communities. One approach is to minimize the self-defined temporal cost of communities between snapshots. [164] proved that this problem is NP-hard and then several works [164, 163, 109] proposed approximation algorithms. However, these algorithms only scale to graphs with thousands of nodes. [158] and [87] propose dynamic community detection al-

gorithms that scale to graphs with hundreds of thousands of nodes. The drawback of [158] is that it cannot track individual community evolution.

The other approach is to match communities detected by static community detection algorithms across temporal snapshots. [65] maps communities between snapshots if their similarity is higher than a threshold. [10, 161] tracks communities between snapshots based on critical community events. These algorithms do not consider any temporal correlation when detecting communities between snapshots.

Self-similarity Measurements and Models Self-similarity describes the phenomenon where a certain property of an object is preserved with respect to scaling in space and/or time. That is, if an object is self-similar, its parts, when magnified, resemble the shape of the whole [139]. Previous works have examined *structural self-similarity* [155, 86], which is concerned with the scale-invariance of certain aspects of the spatial structure of a graph (e.g., node degree distribution) under coarse-graining of vertices. In this work, we focus on *temporal self-similarity*, which describes the scaling properties of certain statistics (e.g., variance, R/S, wavelet coefficients, finite-dimensional distributions) of a time series when computed at different time scales [139]. Note that throughout the chapter we simply refer to temporal self-similarity as self-similarity. We also recall that while for discrete- and continuous-time stochastic processes, the concept of self-similarity is well-defined and has resulted in a large body of literature, for graph structures, the concept of self-similarity remains ill-defined and often reduces to the above-mentioned simple notion of structural self-similarity.

Temporal self-similarity has been discovered in diverse contexts such as ecology, life sciences and stock markets [51], and was first introduced to network traffic for the purpose of modeling the bursty characteristics observed in measured Ethernet LAN traffic [99, 98]. Later studies reported self-similarity also in other network traffic scenarios, including wide-area traffic [140], World Wide Web traffic [39], variable-bit-rate video [23, 59], blog posts [62], messages [148] and emails [51] in communication networks. Note that several of these empirical studies mentioned that in practice, self-similarity is typically observed over a finite range of time scales [1, 59, 66] and can be difficult to discern at both very small and very large time scales.

The traditional way to quantify the existence and degree of self-similarity is to estimate the Hurst parameter H. Four of the most well-known methods for estimating H are the variance analysis [98, 140], R/S analysis [98, 66], Whittle's method [175], and wavelet-based method [1]. Variance and R/S analysis are popular heuristic methods and have been used to measure self-similarity in various areas [23, 39, 59, 66, 98, 140]. However, due to their heuristic nature, they can give incorrect estimates when non-stationary effects in the form of shifts in the mean or slow trends exist in data [83, 166]. Under appropriate conditions, Whittle's method and wavelet-based techniques are more advanced and robust and can be used to obtain more reliable H-estimates with associated confidence intervals.

Generally speaking, there are two classes of self-similar models. The first includes purely mathematical models, *e.g.* fractional Gaussian noise [116], fractional Brownian motion (FBM) [116], fractional ARIMA processes [75] and b-model [173]. However, these models are strictly descriptive and cannot explain the root cause underlying the formation of self-similarity. The second class of models

intends to provide physical reasons behind the existence of self-similarity. Inspired by the application of the class of renewal reward process in economics [165], the superposition of many ON/OFF sources [178, 66] captures the observed self-similar nature of Ethernet LAN traffic, provided the duration of the ON- or OFF-periods have a heavy-tailed distribution. The $M|G|\infty$ queuing model [38, 140, 139], also known as the immigration death model, where sources arrive according to a Poisson process and each source is active for a duration that is described by a heavy-tailed distribution, can also successfully explain self-similar phenomena.

Graph Models In general, graph models can be classified as static graph models or dynamic graph models. We can further classify static models into three sets. One set includes feature-driven models designed to capture one or more static graph features, *e.g.* small-world [174], power-law degree distribution [18], and high clustering coefficients [73]. A second set includes intent-driven models that try to explain the underlying process of graph formation. Nearest neighbor models [168, 167], random walk models [26, 168] and copying models [92, 168] belong to this set. Finally, a third set of models generates graphs based on graph structural statistics instead of graph features. Kronecker graphs [103] apply Kronecker multiplication to generate graphs similar to real graphs. The dK-series model [115] uses subgraph degree distributions to capture increasingly detailed representations of graph structures. Finally, [149] proposes a general technique to produce "realistic" synthetic graphs by calibrating graph models using real graphs.

Dynamic models aim to capture dynamic features of graphs. [104] proposes a Forest Fire model to capture graph densification and diameter shrinking proper-

ties in networks. A later model [118] captures similar properties. The dynamic copying model captures the property of decreasing clustering coefficients, but does not produce a power-law degree distribution [29]. Based on graph structure statistics, [6] proposes a 3D Kronecker model. [5] is a model based on random typing statistics to capture several graph dynamic features. Unlike our work, [5] is not modeled after empirical data of graph dynamics. Finally, [101] designs a model of network evolution, but focuses on reproducing desired structural properties in the final snapshot.

3.7 Summary

This chapter first presents a detailed analysis of user dynamics in the Renren network, a dataset that covers the creation of 19 million users and 199 million edges over a 25-month period. More specifically, we focus on analyzing the network dynamics at different levels of network scale, and identifying the existence of self-similar properties in edge creation process.

Our analysis on dynamics at different scales produced three important findings. First, at the individual node level, unlike the assumption of most graph models, we found that edge creation is increasingly driven by existing nodes as network matures and keeps growing. At the same time, however, the strength of preferential attachment gradually weakens over time as the network grows and matures. Second, at the community level, we use the incremental Louvain algorithm to track communities across snapshots, and show users' activities are in fact impacted by the users in the same community. Finally, we analyze the dynamics following the unique network merge event in detail, and find that due to the large

number of new nodes joining the network, the impact of this event fades quickly after a short term.

To detect self-similarity, we use a range of techniques, including R/S analysis, the variance fitting method, and a wavelet-based method. Given the presence of the diurnal pattern in long-term time scales, from days to weeks, we rely on the robust wavelet-based method to not only reliably identify the existence self-similarity, but also determine the time scales where self-similarity is visible, *i.e* 3 hours.

Motivated by the results of a detailed analysis of dynamics in the Renren network, we propose a complete dynamic graph model for large online social networks. Our model includes a temporal component that defines when and how many new edges are formed across all the users, and a spatial component that defines where in the graph new edges form. While the temporal component captures the co-existence of a diurnal pattern and self-similar behavior, the spatial component ensures that the edge creation process is primarily driven by existing nodes and is responsible for the dynamic properties of the graph structure, such as graph densification, shrinking network diameter, and decreasing local clustering. Our extensive validation shows that our model accurately captures both the temporal and spatial dynamic properties of the Renren network. Importantly, this model produces a sequence of time-stamped edge creation events, which uniquely define the formation and evolution of a social network in time and space, and can be used by other researchers for generating "realistic" dynamic graph traces.

Chapter 4

Privacy Preserving Graph-Sharing

4.1 Introduction

[1]Studying structure of real social and computer networks through graph analysis can produce insights on fundamental processes such as information dissemination, viral spread and epidemics, network dynamics and resilience to attacks [78, 91, 135, 8]. The use of real graphs generated from measurement data is invaluable, and can be used to validate theoretical models or realistically predict the effectiveness of applications and protocols [3, 34, 149, 179].

Unfortunately, there is often a direct tension between the need to distribute real network graphs to the research community, and the privacy concerns of users or entities described by the dataset. For example, social graphs from real measurements are used to capture a variety of artifacts in online social networks, including strength of social ties, number and frequency of social interactions, and flow of information. Similarly, detailed topology graphs of enterprise networks or major

[1]Abbreviated version of content in this chapter can be found in paper "Sharing Graphs Using Differentially Private Graph Models" [150].

ISPs contain confidential information about the performance and robustness of these networks. Releasing such sensitive datasets for research has been challenging. Despite the best of intentions, researchers often inadvertently release more data than they originally intended [126, 127, 192]. Past experience has taught us that traditional anonymization techniques provide limited protection, and often can be overcome by privacy attacks that "de-anonymize" datasets using external or public datasets [13, 126, 127].

Thus we are left asking the question, *how can researchers safely share realistic graph datasets from measurements without compromising privacy?* One option is to develop and apply stronger anonymization techniques [71, 111], many of which modify the graph structure in subtle ways that improve privacy but retain much of the original graph structure. However, these approaches generally only provide resistance against a specific type of attack, and cannot provide protection against newly developed deanonymization techniques. Techniques exist in the context of databases and data mining, which provide provable levels of protection [55, 56], but are not easily applied to graphs. Still other techniques can protect privacy on graphs, but must significantly change the graph structure in the process [142, 71].

Our approach to provide graph privacy and preserve graph structure. We seek a solution to address the above question, by starting with observation that any system for sharing graphs must deal with the tension between two goals: *protecting privacy* and *achieving structural similarity to the original, unmodified graph.* At one extreme, we can distribute graphs that are isomorphic to the original, but vulnerable to basic deanonymization attacks. At the other extreme, we can distribute random graphs that share no structural similarities to the origi-

nal. These graphs will not yield any meaningful information to privacy attacks, but they are also not useful to researchers, because they share none of the real structures of the original graph.

Ideally, we want a system that can produce graphs that span the entire privacy versus similarity spectrum. In such a system, users can specify a desired level of privacy guarantee, and get back a set of graphs that are similar to the real graph in structure, but have enough differences to provide the requested level of privacy.

The main premise of our work is that we can build such a system, by distilling an original graph G into a statistical representation of graph structure, adding controlled levels of "noise," and then generating a new graph G' using the result statistics. This requires two key components. First, we need a way to accurately capture a graph's structure as a set of structural statistics, along with a generator that converts it back into a graph. For this, we use the *dK-series*, a graph model that is capable of capturing sufficient graph structure at multiple granularities to uniquely identify a graph [115, 45]. We can achieve the desired level of privacy by introducing a specific level of noise into G's degree correlation statistics. Second, we need a way to determine the appropriate noise necessary to guarantee a desired level of privacy. For this, we develop new techniques rooted in the concept of ϵ-*differential privacy*, a technique previously used to quantify privacy in the context of statistical databases.

In chapter, we develop *Pygmalion*, a differentially private graph model for generating synthetic graphs. Pygmalion preserves as much of the original graph structure as possible, while injecting enough structural noise to guarantee a chosen level of privacy against privacy attacks. Initially, we formulate a basic differentially private graph model, which integrates controlled noise into the dK degree

distributions of an original graph. We use the dK-2 series, which captures the frequency of adjacent node pairs with different degree combinations as a sequence of frequency values. However, when we derive the necessary conditions required to achieve ϵ-differential privacy, they show that an asymptotical bound for the required noise grows polynomially with the maximum degree in the graph. Given the impact of dK values on graph structure, these large noise values result in synthetic graphs that bear little resemblance to the original graph.

To solve this challenge, we seek a more accurate graph model by significantly reducing the noise required to obtain ϵ-differential privacy. We develop an algorithm to partition the statistical representation of the graph into clusters, and prove that by achieving ϵ-differential privacy in each cluster, we achieve the same property over the entire dataset. Using a degree-based clustering algorithm, we reduce the variance of degree values in each cluster, thereby dramatically reducing the noise necessary for ϵ-differential privacy. Finally, we apply isotonic regression [19] as a final optimization to further reduce the effective error by more evenly distributing the added noise.

We apply our models to a number of Internet and Facebook graphs ranging from 14K nodes to 1.7 million nodes. The results show that for a given level of privacy, our degree-based clustering algorithm reduces the necessary noise level by *one order of magnitude*. Isotonic regression further reduces the observed error in dK values on our graphs by 50%. Finally, we experimentally show that for moderate privacy guarantees, synthetic graphs generated by Pygmalion closely match the original graph in both standard graph metrics and application-level experiments.

Access to realistic graph datasets is critical to continuing research in both social and computer networks. Our work shows that differentially-private graph models are feasible, and Pygmalion is a first step towards graph sharing systems that provide strong privacy protection while preserving graph structures.

4.2 A differential private graph model

In this section, we provide background on graph anonymization techniques, and motivate the basic design of our approach to graph anonymization. First, we discuss prior work, the inherent challenges in performing graph anonymization, potential privacy risk in generating synthetic graphs, and our desired privacy goals. Second, we introduce the main concepts of ϵ-Differential Privacy, and lay out the preconditions and challenges in leveraging this technique to anonymize graphs. Finally, we motivate the selection of the dK-series as the appropriate graph model on which to build our system.

4.2.1 Background and Goals

A significant amount of prior work has been done on protecting privacy of datasets. We summarize them here, and clarify our privacy goals in this project.

Private Datasets. Many research efforts have developed privacy mechanisms to secure large datasets. Most of these techniques, including cryptographic approaches [20] and statistical perturbations [134, 56], are designed to protect structured data such as relational databases, and are not applicable to graph datasets. An alternative, probabilistic approach to privacy is k-anonymity [160]. It is de-

signed to secure sensitive entries in a table by modifying the table such that each row has at least $k-1$ other rows that are identical [55]. Several public datasets have been successfully anonymized with k-anonymity [121, 2] or through clustering-based anonymization strategies [24].

Graph Anonymization. Several graph anonymization techniques have been proposed to enable public release of graphs without compromising user privacy. Generally, these techniques only protect against specific, known attacks. The primary goal of these anonymization techniques is to prevent attackers from identifying a user or a link between users based on the graph structure. Several anonymization techniques [71, 191, 111, 193, 142] leverage the k-anonymity model to create either k identical neighborhoods, or k identical-degree nodes in a target graph. These types of "attack-specific" defenses have two significant limitations. First, recent results have repeatedly demonstrated that researchers or attackers can invent novel, unanticipated de-anonymization attacks that destroy previously established privacy guarantees [126, 127, 13, 189]. Second, many of these defenses require modifications to the protected graph that significantly alter its structure in detectable and meaningful ways [71, 142].

Graph Models. Because of high privacy risk in sharing real graphs, an attractive alternative to protecting graph privacy is to generate synthetic graphs with near identical graph properties matching real graphs. Along this direction, two models are proposed to capture structural characteristics of real graphs. One is Kronecker graph model [103], which uses Kronecker multiplication to approximate real graph structures. The other model is called dK-graph model, which extracts

subgraph degree distributions and reproduces synthetic graphs with identical degree distributions. However, the high computational complexity limits their accuracy in practical settings. Alternatively, [149] proposes a systematic approach to calibrate graph models with real graphs. Given a real graph and a model, this approach finds the best parameters for the model by adaptively searching the parameter space and looking for the set of parameters with minimum structural differences between the generated graph and the real graph. The structural difference is quantified as the Euclidean distances between the dK-series of the two graphs. By applying the method, the improved versions of traditional graph models, such as preferential attachment [18], nearest neighbor model [168], and forest fire model [104], can produce synthetic graphs similar to the original real graphs in term of structures. However, what would happen if the models are highly accurate such that the generated graphs are identical with the real graphs? Clearly, privacy problem will remain in sharing the accurate synthetic graphs. Thus, a new graph model with privacy guarantee is needed.

Our Goals: Edge vs. Node Privacy. In the context of privacy for graphs, we can choose to focus on protecting the privacy of either node or edges. As will become clear later in this chapter, our approach of using degree correlations (*i.e.* the dK-series), captures graph structure in terms of different subgraph sizes, ranging from 2 nodes connected by a single edge (dK-2) to larger subgraphs of size K.

Our general approach is to produce synthetic graphs by adding controlled perturbations to the graph structure of the original graph. This approach can provide protection for both node privacy and edge privacy. This choice directly

impacts the sensitivity of the graph privacy function, and as a result, how much structural noise must be introduced to obtain a given level of privacy guarantees.

In this chapter, we choose to focus on edge privacy as our goal, and apply this assumption in our analysis of our differential privacy system in Section 4.3 and 4.4. We chose to target edge privacy because our work was originally motivated by privacy concerns in sharing social graphs, where providing edge privacy would address a number of practical privacy attacks.

4.2.2 Differential Privacy

Our goal is to create a novel system for the generation of anonymized graphs that support two key properties:

1. Provides quantifiable privacy guarantees for graph data that are "future-proof" against novel attacks.

2. Preserves as much original graph structure as possible, to ensure that anonymized data is still useful to researchers.

Differential privacy [47] is a recently developed technique designed to provide and quantify privacy guarantees in the context of statistical databases [48, 72]. Others have demonstrated the versatility of this technique by applying differential privacy to distributed systems [146], network trace anonymization [120], data compression techniques [180], and discrete optimization algorithms [69]. Other work focused specifically on applying differential privacy to simple graph structures such as degree distributions [70, 72]. In contrast, our work has the potential to inject changes at different granularities of substructures in the graph, instead of focusing on a single graph metric.

One piece of prior work tried to guarantee graph privacy by adding differential privacy to Kronecker graphs [122]. Whereas this approach tries to guarantee privacy by perturbing the Kronecker model parameters, our strategy acts directly on graph structures, which provides tighter control over the perturbation process. Unfortunately, the author asserts there are incorrect results in the paper[2].

Basic Differential Privacy. The core privacy properties in differential privacy are derived from the ability to produce a query output Q from a database D, which could also have been produced from a slightly different database D', referred to as D's neighbor [47].

Definition 1. *Given a database D, its neighbor database D' differs from D in only one element.*

We obtain differential privacy guarantees by injecting a controlled level of statistical noise into D [49]. The injected noise is calibrated based on the sensitivity of the query that is being executed, as well as the statistical properties of the Laplace stochastic process [50]. The *sensitivity* of a query is quantified as the maximum amount of change to the query's output when one database element is modified, added, or removed. Together, query sensitivity and the ϵ value determine the amount of noise that must be injected into the query output in order to provide ϵ-differential privacy.

Differential privacy works best with *insensitive queries*, since higher sensitivity means more noise must be introduced to attain a given desired level of privacy. Thus insensitive queries introduce lower levels of errors, and provide more accurate query results.

[2]See the author's homepage.

4.2.3 Differential Privacy on Graphs

We face two key challenges in applying differential privacy concepts to privacy protection on graphs. First, we must determine a "query" function in our context, which we can use to apply differential privacy concepts. Second, the sensitivity of this query function must be low enough, so that we can attain privacy guarantees by introducing only low levels of noise, thus allowing us to preserve the accuracy of the results. In our context, this means that we want to generate graphs that retain the structure and salient properties of the original graph. We address the former question in this section by proposing the use of the dK-series as our graph query operation. We address the accuracy question in Sections 4.3 and 4.4, after fully explaining the details of our system.

Recall that the problem we seek to address is to anonymize graph datasets so that they can be safely distributed amongst the research community. We leverage a *non-interactive* query model [47], such that the original graph structure is queried only once and the entire budget to enforce privacy is used at this time. dK is used to query the graph and the resulting dK-series is perturbed under the differential privacy framework. Note that only the differentially private dK-series is publicized. Unlike applications of differential privacy in other contexts, we can now generate multiple graphs using this differentially private dK-series without disrupting the level of privacy of the original graph. Therefore, we use a non-interactive query model to safely distributed graph datasets without being constrained to a single dataset.

The dK-Graph Model. We observe that the requirements of this query function can be met by a descriptive graph model that can transform a graph into a set

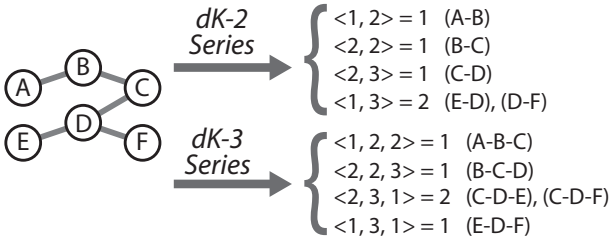


Figure 4.1: An illustrative example of the dK-series. The dK-2 series captures the number of 2-node subgraphs with a specific combination of node-degrees, and the dK-3 captures the number of 3-node subgraphs with distinct node-degree combinations.

of structural statistics, which are then used to generate a graph with structure similar to the original. Specifically, we propose to use the dK-graph model [115] and its statistical series as our query function. dK captures the structure of a graph at different levels of detail into statistics called dK-series. dK can analyze an original graph to produce a corresponding dK-series, then use a matching generator to output a synthetic graph using the dK-series values as input. The dK-series is the degree distribution of connected components of some size K within a target graph. For example, dK-1 captures the number of nodes with each degree value, *i.e.* the node degree distribution. dK-2 captures the number of 2-node subgraphs with different combinations of node degrees, *i.e.* the joint degree distribution. dK-3 captures the number of 3-node subgraphs with different node degree combinations, *i.e.* an alternative representation of the clustering coefficient distribution. dK-n (where n is the number of nodes in the graph) captures the complete graph structure. We show a detailed example in Figure 4.1, where we list dK-2 and dK-3 distributions for a graph.

dK is ideal for us because the dK-series is a set of data tuples that provides a natural fit for injecting statistical noise to attain differential privacy. In addition, together with their matching generators, higher levels of dK-series, *i.e.* $n > 3$, could potentially provide us with a bidirectional transformation from a graph to its statistical representation and back.

While larger values of K will capture more structural information and produce higher fidelity synthetic graphs, it comes at the expense of higher computation and storage overheads. Our work focuses on the dK-2 series, because generator algorithms have not yet been discovered for dK-series where $K \geq 3$. While this may limit the accuracy of our current model, our methodology is general, and can be used with higher order dK-series when their generators are discovered.

ϵ-Differential Privacy in Graphs. Given the above, we can now outline how to integrate differential privacy in the context of graphs. An ϵ-differentially private graph system would output a graph that given a statistical description of an input graph, the probability of seeing two similar graphs as the real input graph is close, where closeness between the two probabilities is quantified by ϵ. A larger value of ϵ means it is easier to identify the source of the graph structure, which means a lower level of graph privacy.

Prior work has demonstrated that in many cases, accuracy of query results on differentially private databases can be improved by decomposing complex queries into sequences of "simple counting queries" that happen to have extremely low sensitivity [27, 28, 48]. Unfortunately, this approach will not work in our context, since our goal is to achieve privacy guarantees on whole graph datasets, and not just privacy for simple graph queries such as node degree distributions. In the

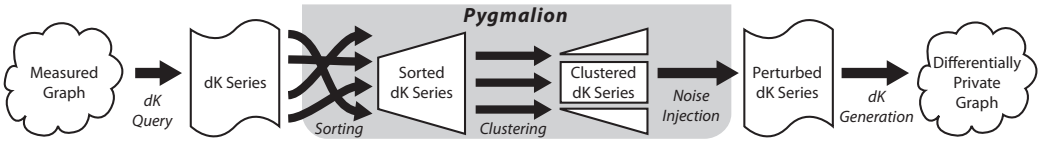


Figure 4.2: Overview of Pygmalion. ϵ-differential privacy is added to measured graphs after sorting and clustering the dK-2-series.

next section, we start with a basic formulation of a differentially private graph model, and then provide an optimized version. We illustrate the final process, shown as *Pygmalion* in Figure 4.2.

4.3 Basic Design

In this section, we perform the analytical steps necessary to integrate ϵ-differential privacy into the dK graph model. Our goal is to derive the amount of noise necessary to achieve a given ϵ-privacy level. The amount of Laplacian noise necessary is a function of both ϵ, the user-specified privacy parameter, and S, the sensitivity of the dK function. First, we formally define the dK-2 series, and derive its sensitivity S_{dK-2}. Next, we describe the dK-perturbation algorithm (dK-PA) for injecting noise into the original dK-2 series, and prove that it provides the desired ϵ-differential privacy. Our analysis shows that the asymptotic bound on noise used in dK-PA grows polynomially with maximum node degree, which means we need to inject relatively large levels of noise to guarantee ϵ-privacy. Finally, as expected, our experiments on real graphs confirm that dK-PA generates synthetic graphs with significant loss in accuracy. This poor result motivates our search for improved techniques in Section 4.4.

4.3.1 Sensitivity of $dK-2$

dK-function. We formally define dK-2 as a function over a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges connecting pair of nodes in V:

$$dK(G) : G^n \rightarrow \Im$$

where G^n is the set of graphs with $n = |V|$ nodes and $\Im$ is the set of unique degree tuples in the dK-2-series with the corresponding count of instances in G. Formally, $\Im$ is a collection of $\{d_x, d_y; k\}$ where each entry represents that the number of connected components of size 2 with degree (d_x, d_y) is k. Let m be the cardinality of $\Im$. Because the maximum number of entries in dK-2 is bounded by the number of possible degree pairs, $\sum_{i=1}^{d_{max}} i$, where d_{max} be the maximum node degree in G, thus $m = O(d_{max}^2)$. Prior studies have demonstrated that in large network graphs d_{max} is upper bounded by $O(\sqrt{n})$ [179, 94], and thus, in those cases, m is upper bounded by $O(n)$.

Sensitivity Analysis. In the context of differential privacy, the sensitivity of a function is defined as the maximum difference in function output when one single element in the function domain is modified. The domain of dK-2 is a graph G. Neighbor graphs of G are all the graphs G' which differ from G by at most a single edge. Changing a single edge in G will result in one or more entries changing in the corresponding dK-2-series. Thus, the sensitivity of dK-2 is computed as the maximum number of changes in the dK-2-series among all of G's neighbor graphs.

Lemma 1. *The sensitivity of dK-2 on a graph G, S_{dK-2}, is upper bounded by* $4 \cdot d_{max} + 1$.

Proof. Let e be a new edge added to a graph $G = (V, E)$ between any two nodes $u, v \in V$. Once the edge e is added to G the degrees of u and v increase from d to $(d + 1)$ and from d' to $(d' + 1)$ respectively. This graph transformation produces the following changes in the dK-2 on G: the frequency k of tuple $\{d+1, d'+1; k\}$ gets incremented by 1 because of the new edge (u, v). For example, a new edge between A and C in Figure 4.1 produces an increment of the frequency k of the tuple $\{2, 3; k\}$ from $k = 1$ to $k = 2$. Furthermore, a total of $d + d'$ already present tuples need to be updated with the new degree of u and v, and so the tuples with the old degrees get decremented by a total of $d + d'$ and the tuples reflecting the new degree get incremented for a total of $d + d'$. To summarize, the overall number of changes in the dK-2 -series is $2(d + d') + 1$. In the worst case, when u and v are nodes of maximum degree d_{max}, the total number of changes in the original dK-2-series by adding an edge between u and v is upper bounded by $4 \cdot d_{max} + 1$. □

Lemma 1 derives only the upper bound of the sensitivity because, as in Definition 3 [47], it is the sufficient condition to derive the necessary amount of noise to achieve a given ϵ-privacy level. Lemma 1 shows that the sensitivity of dK-2 is high, since d_{max} has been shown to be $O(\sqrt{n})$ in measured graphs [179, 94]. Note that prior work on differential privacy [27, 28, 48, 70] generally involved functions with a much lower sensitivity, *i.e.* 1. In these cases, the low sensitivity means that the amount of noise required to generate differentially private results is very small. In contrast, the sensitivity of our function indicates that the amount of noise needed to guarantee ϵ-differential privacy in dK-2 will be high. Therefore, the accuracy of synthetic graphs generated using this method will be low. Note

that if we use a higher order dK-series, *i.e.* $K \geq 3$, we would have found an even higher sensitivity value, which may further degrade the accuracy of the resulting synthetic graphs.

4.3.2 The dK-Perturbation Algorithm

We now introduce the dK-perturbation algorithm (dK-PA) that computes the noise to be injected into dK-2 to obtain ϵ-differential privacy [47]. In dK-PA, each element of the dK-2-series is altered based on a stochastic variable drawn from the Laplace distribution, $Lap(\lambda)$. This distribution has density function proportional to $e^{-\frac{|x|}{\lambda}}$, with mean 0 and variance $2\lambda^2$. The following theorem proves the conditions under which ϵ-differential privacy is guaranteed [50].

Theorem 1. *Let $\widetilde{DK}$ be the privacy mechanism performed on dK such that $\widetilde{DK(G)} = dK(G) + Lap(\frac{S_{dK-2}}{\epsilon})^m$. For any G and G' differing by at most one edge, $\widetilde{DK}$ provides ϵ-differential privacy if:*

$$\Big| \ln \frac{Pr[\widetilde{DK(G)} = s]}{Pr[\widetilde{DK(G')} = s]} \Big| \leq \epsilon$$

Proof. Let $s =< s_1, s_2, ..., s_m >$ be a possible output of $\widetilde{DK(G)}$ and m the number of its entries, and let G' be the graph with at most one different edge from G. Using the conditional probabilities, we have:

$$\frac{Pr[\widetilde{DK(G)} = s]}{Pr[\widetilde{DK(G')} = s]} = \prod_{i=1}^{m} \frac{Pr[\widetilde{DK(G)}_i = s_i | s_1, ...s_{i-1}]}{Pr[\widetilde{DK(G')}_i = s_i | s_1, ...s_{i-1}]},$$

since each item of the product has the first $i-1$ values of dK-2 fixed. Each s_i is the result of applying Laplacian noise calibrated by S_{dK-2}. Note that Lemma 1 has

studied the sensitivity of dK-2, S_{dK-2}, under the condition that two graphs differ by at most one edge. Thus, the conditional probability is Laplacian, allowing us to derive the following inequalities:

$$\prod_{i=1}^{m} \frac{Pr[\widetilde{DK(G)}_i = s_i | s_1, ...s_{i-1}]}{Pr[\widetilde{DK(G')}_i = s_i | s_1, ...s_{i-1}]} \leq \prod_{i=1}^{m} e^{\frac{|\widetilde{DK(G)}_i - \widetilde{DK(G')}_i|}{\sigma}}$$

where σ is the scale parameter of the Laplace distribution that is $\frac{4d_{max}+1}{\epsilon}$. Thus,

$$\prod_{i=1}^{m} e^{\frac{|\widetilde{DK(G)}_i - \widetilde{DK(G')}_i|}{\sigma}} = e^{\frac{||\widetilde{DK(G)} - \widetilde{DK(G')}||_1}{\sigma}}$$

where, by definition $\widetilde{DK(G)} = dK(G) + Lap(\frac{S_{dK-2}}{\epsilon})$, and $||DK(G) - DK(G')||_1 \leq S_{dK-2}$ with $S_{dK-2} \leq 4d_{max} + 1$ as proved in Lemma 1. Thus, we have:

$$e^{\frac{||\widetilde{DK(G)} - \widetilde{DK(G')}||_1}{\sigma}} =$$

$$= e^{\frac{||dK(G) + Lap(\frac{S_{dK-2}}{\epsilon}) - dK(G') - Lap(\frac{S_{dK-2}}{\epsilon})||_1}{\sigma}} \leq e^{\frac{\frac{4d_{max}+1}{4d_{max}+1}}{\epsilon}} = e^{\epsilon}$$

and so, by applying the logarithmic function, we have that

$$\Big| \ln \frac{Pr[\widetilde{DK(G)} = s]}{Pr[\widetilde{DK(G')} = s]} \Big| \leq \epsilon$$

which concludes the proof. □

Theorem 1 shows that by adding noise to the dK-2-series using independent Laplace random variables calibrated by S_{dK-2} from Lemma 1, we achieve the desired ϵ-privacy.

Quantifying Accuracy. We apply the *error analysis* proposed by [72] on dK-PA to quantify the accuracy of the synthetic graphs it produces, compared to the original graphs.

Definition 2. *For a perturbed dK-2-series that is generated by the privacy mechanism $\widetilde{DK}$ on a graph G, as defined in Theorem 1, the estimated error on $\widetilde{DK}$ can be computed as the expected randomization in generating $\widetilde{DK}$.*

We now quantify the expected randomization in $\widetilde{DK}$:

$$\sum_{i=1}^{m} E[(\widetilde{DK(G)}_i - dK(G)_i)^2] = mE[Lap(\frac{S_{dK-2}}{\epsilon})^2]$$

Using Lemma 1 and that $m = O(d_{max}^2)$we have:

$$mE[Lap(\frac{S_{dK-2}}{\epsilon})^2] = mVar(Lap(\frac{d_{max}}{\epsilon})) = \frac{2m \cdot d_{max}^2}{\epsilon^2} = O(\frac{d_{max}^4}{\epsilon^2}).$$

This asymptotical bound shows that the noise injected by dK-PA into dK-2 scales with the fourth-degree polynomial of d_{max}. This result implies that synthetic graphs generated by dK-PA will have relatively low accuracy because of the large error introduced by the perturbation process. Furthermore, it implies that even for relatively weak privacy guarantees, dK-PA will introduce large errors that may significantly change the structure of the resulting synthetic graphs from the original.

4.3.3 Validation on Real Graphs

At this point, we have demonstrated analytically that the impact of adding noise to the dK-2-series using dK-PA will result in synthetic graphs that deviate

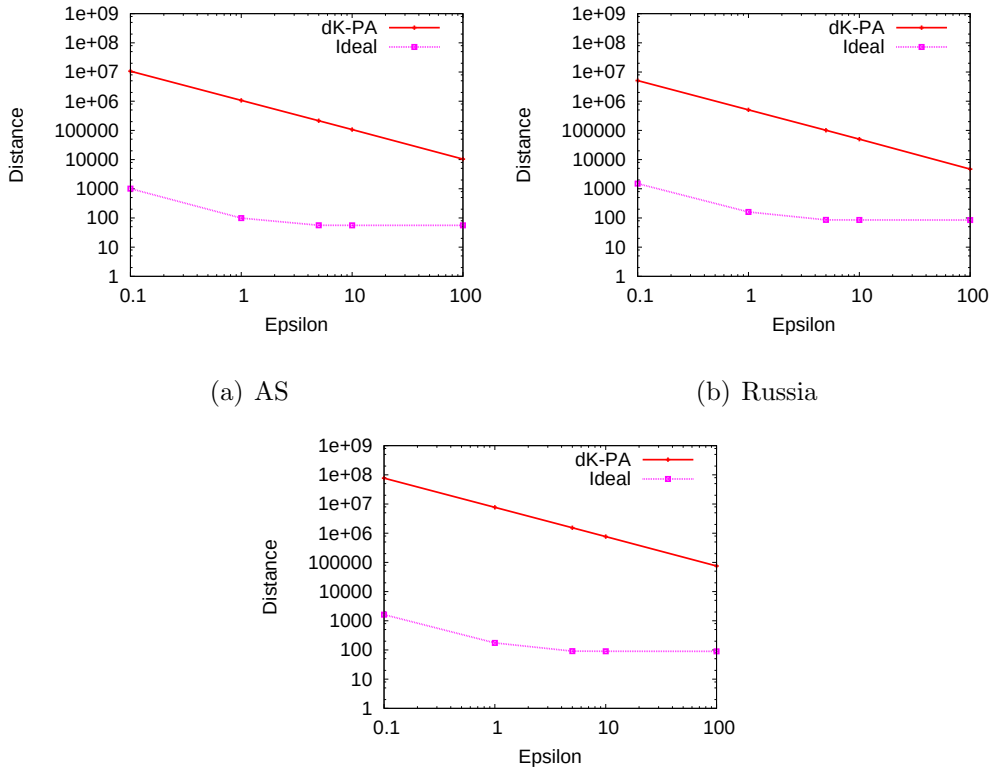
Type	Graph	Nodes	Edges
Internet	WWW	325,729	1,090,108
	AS	16,573	40,927
Facebook	Monterey Bay	14,260	93,291
	Russia	97,134	289,324
	Mexico	598,140	4,552,493
	LA	603,834	7,676,486

Table 4.1: Different measurement graphs used for experimental evaluation.

significantly from the originals. In this section, we empirically evaluate the impact of adding noise to the dK-2-series by executing dK-PA on real graphs.

Methodology. To illustrate that our system is applicable to different types of graphs, we select a group of graphs that include social graphs from Facebook [179, 149], a WWW graph [7] and an AS topology graph [135] crawled on Jan 1st, 2004, which have been used in prior graph mining studies [93]. The social graphs were gathered using a snowball crawl of the Facebook regional networks [179], and show graph metrics highly consistent with Facebook graphs generated using unbiased sampling techniques [61]. Table 4.1 lists the graphs used in our evaluation, which range from 14K nodes to 650K nodes.

We extract the dK-2-series for each graph, introduce noise using the dK-PA strategy, then compute the Euclidean distance between the perturbed dK-2-series and the original as a measure of the level of graph structural error introduced. We computed results for all graphs in Table 4.1, and they are consistent. For brevity, we limit ourselves to report results only for the AS graph, the WWW graph, and the Russia Facebook graph. We choose Russia to represent our social graphs because its results are representative of the other graphs, and its size does not result in extremely long run time for our experiments.



(a) AS

(b) Russia

(c) WWW

Figure 4.3: The noise required for different privacy levels quantified as the Euclidean distance between a graph's original and perturbed dK-2 series.

Results. Figure 4.3 shows that the dK-PA strategy produces a large error for small values of ϵ (*i.e.* strong privacy guarantees). We compute the error as the Euclidean distance between the original dK-2-series and the perturbed dK-2-series with dK-PA strategy. As we mentioned, the low level of accuracy is due to the large noise dK-PA injects into dK-2, resulting in a perturbed dK-2 that is significantly different from the original. The bright side is that the dK-PA strategy is robust across different datasets, and the error decreases exponentially

as ϵ grows, which is shown by the linear correlation in the log-log scale plot of Figure 4.3.

The high error is largely due to the high sensitivity of our function dK-2. To understand the potential lower-bound on the error, we imagine a scenario where if we had a function with sensitivity of 1, then we could achieve much lower error, plotted in Figure 4.3 as the *Ideal* line. Note that this line is a hypothetical lower bound that is only meant to demonstrate the impact of the dK function's sensitivity on the final result. Indeed, Figure 4.3 shows that the loss in accuracy of our model can largely be attributed to the sensitivity of the dK-2 series.

4.4 Improvement via Partitioning

The results in the previous section demonstrate the loss of accuracy in the perturbed dK-2-series after adding noise to guarantee ϵ-differential privacy. In this section we propose a novel algorithm called Divide Randomize and Conquer (DRC) that enables more granular control over the noise injected into the dK-2-series. This qualifies DRC to support ϵ-differential privacy while also allowing for more accurate results. First, we discuss the design of DRC and prove that it does guarantee ϵ-differential privacy. Next, we investigate the amount of error introduced with this approach, and show that DRC requires significantly less noise than dK-PA to achieve an equal level of privacy. Finally, we propose an optimized version of DRC, called LDRC, and empirically verify the improved accuracy of our algorithms using measured graphs.

4.4.1 Divide Randomize and Conquer Algorithm

Our goal is to develop an improved privacy mechanism that significantly reduces the amount of noise that must be added to achieve a given level of ϵ-privacy. While we cannot change the fact that the sensitivity of dK-2 scales with d_{max}, our insight is to partition data in the dK-2-series into a set of small sub-series, then apply the perturbation independently to achieve ϵ-privacy within each sub-series.

If we carefully perform the partitioning to group together tuples with similar degree, we effectively reduce the value of d_{max} for each of the vast majority of sub-series. This means we can achieve ϵ-privacy on each sub-series for a fraction of the noise required to achieve ϵ-privacy across the entire series. We will then prove that ϵ-differential privacy holds across the entire dK-2-series if it holds for each of the partitioned sub-series. Thus, we produce an alternative algorithm that achieves the same level of privacy as dK-PA, while introducing significantly less noise.

We instantiate our ideas as the Divide Randomize and Conquer algorithm (DRC). The core steps of DRC are:

1. Partition (*Divide*) the dK-2-series into sub-series with specific properties;
2. Inject noise into each sub-series (*Randomize*);
3. *Conquer* the perturbed sub-series into a single dK-2-series.

In the remainder of this section we discuss the partitioning step of DRC. We first define an ordering function on dK-2 to sort tuples with similar sensitivity. The ordered dK-2 is then partitioned into contiguous and mutually disjoint sub-series. We prove that the properties of these sub-series lead to the definition

of a novel sensitivity function and consequently to a novel methodology to add noise. Noise injection, conquering, and the resulting error analysis are discussed in Section 4.4.2.

∂ ordering on dK-2. The dK-2-series is sorted by grouping dK-tuples with numerically close pairs of degrees. In particular, the dK-tuples are sorted in the new dK-2 series, named β-series, by iteratively selecting from the original series all the tuples $\{d_x, d_y; k\}$ with degrees $(d_x \ \& \ d_y) \leq i, \ \forall \ i \in [1, d_{max}]$. Thus, the β-series is simply the sorted list of dK-tuples that adhere to the above inequality ordering. For example, the tuple $\{1, 2; k\}$ is closer to $\{5, 5; k'\}$ than to $\{1, 8; k''\}$. We can formally describe this transformation with the following function:

Definition 3. *Let ∂ be the sorting function on dK-2 which is formally expressed as:*

$$\partial(i) = \min_{d_x, d_y \in dK} \{\max(d_x, d_y) \geq \max(d_{x'}, d_{y'}) = \partial(i-1) \ \}$$

Note that $\{d_x, d_y; k\} \neq$ the first $i-1$ tuples. Thus, the ∂ function is a transformation of dK-2 such that $\partial : \Im \rightarrow \beta$ where β identifies the ordered dK-2.

Partitioning the β-Series. The β-series is partitioned into $\widetilde{m}$ sub-series, with the ith named β_i for $i \in [1, \widetilde{m}]$. The partition of β is based on two properties. First, the ∂ ordering has to be obeyed and thus each partition can only acquire *contiguous* tuples in the β-series. Second, each tuple can appear in *one and only one* sub-series. Given the ∂ ordering and the above two rules we can guarantee *mutually disjoint* and *contiguous* sub-series β_i. These two constraints are fundamental to satisfying the sensitivity properties we prove in the following Lemma 2 and Lemma 3.

Sensitivity of β_i sub-series. The sensitivity of each β_i-series can be studied following the same logic used to find the sensitivity of dK-2, by quantifying the maximum number of changes that may occur in the β_i-series due to an edge change in the graph G. Due to the ∂ ordering imposed in each sub-series, we can show that the maximum degree in each β_i plays a fundamental role in bounding its sensitivity.

Lemma 2. *The sensitivity S_{β_i} of a sub-series β_i with tuple degrees almost equal to $d_k + 1$ is upper bounded by $4 \cdot d_k + 1$.*

The proof of this lemma is sketched because it follows the logic of Lemma 1. Due to the proposed ∂ ordering, each sub-series i is composed only of tuples where both degrees are less than or equal to a particular integer d. The worst-case (*i.e.* the maximum number of changes to the tuples in the same β_i) occurs when the tuple with degrees $d - 1$ are in the same sub-series. Therefore, the maximum number of changes occur when a new edge is added between two nodes (u, v) both with degree $d - 1$, after which both nodes u and v have degree d. Adding a new edge between u and v causes $d_k = d - 1$ entries in β_i to become invalid. Each invalid entry is replaced with new entry of degree d. Thus, the upper bound on the total number of changes is $2 \cdot d_k$ deletions, $2 \cdot d_k$ additions, and one new edge, with the total being $4 \cdot d_k + 1$.

Given the partitioning approach and the imposed ∂ ordering across sub-series, we are able to exploit further properties on the β_is-series. In particular, the sensitivity of any β_i is independent from the location where the change occurs in the graph. Conversely, the sensitivity of a particular partition is dependent on the tuple with the highest degree values, as proved in Lemma 2. Therefore:

Lemma 3. *The sensitivity of any β_i is independent by the sensitivity of any other β_j with $i \neq j$.*

Proof. The proof proceeds by contradiction from the following assumption: *the sensitivity of a β_i is impacted by a change occurring in a β_j with $i \neq j$.* Without loss of generality, assume $i < j$, and $\partial(i')$ is a tuple in β_i and $\partial(j')$ is a tuple in β_j, as from Definition 3. Assume that an edge is formed between a node x with corresponding tuples $< \partial(i'), \partial(i'+1), .. > \in \beta_i$ and a node y with corresponding tuples $< \partial(j'), \partial(j'+1)... > \in \beta_j$. The maximum number of changes that can occur due to this event is bounded by the degree values of x and y. Let d be the new degree of x. The maximum number of tuples that can change in β_i are $d-1$ tuples that get deleted and d that get added, which is $< 2 \cdot d$. Symmetrically, let b be the new degree of y so the maximum number of tuples that can change in β_j is $< 2 \cdot b$. Even if d and b are equal to the maximum degree value d_k within their sub-series, as demanded in Lemma 2, the number of changes involved in each sub-series is $2 \cdot d_k < 4 \cdot d_k + 1$ which means that the sensitivity of both β_i and β_j are not mutually effected, which contradicts the hypothesis. □

4.4.2 Theoretical Analysis

This section is devoted to the theoretical analysis of the privacy and accuracy properties the DRC approach achieves. First, we prove that ϵ-differential privacy can be applied to each sub-series created during the partitioning phase of DRC. Next, we build on this result to prove that the individual differentially private sub-series' can be reunified into a complete dK-2-series that is also ϵ-differentially

private. Lastly, we perform error analysis on DRC and compare the results to dK-PA.

Analyzing ϵ-Privacy in β_is. We now quantify the privacy of each β_i and prove that they satisfy ϵ-differential privacy.

Theorem 2. *For each cluster β_i with $i = 1, .., \widetilde{m}$, let $\widehat{\beta}_i$ be a novel privacy mechanism on β_i such that $\widehat{\beta}_i = \beta_i + Lap(\frac{S_{\beta_i}}{\epsilon})^{|\beta_i|}$. Then, for all sub-series β_i and β_i' derived from graphs G and G' that differ by at most one edge, $\widehat{\beta}_i$ satisfies ϵ-differential privacy if:*

$$\Big| \ln \frac{Pr[\widehat{\beta}_i = s]}{Pr[\widehat{\beta}_i' = s]} \Big| \leq \epsilon$$

Proof. Let m^* be the the cardinality of cluster β_i. Let G' be a graph with at most one edge different from G. Let s_j be the j^{th} item of the $\widehat{\beta}_i$-series, that is $\widehat{\beta}_i[j] = s_j$. Using the conditional probability on s_j we can write:

$$\frac{Pr[\widehat{\beta}_i = s]}{Pr[\widehat{\beta}_i' = s]} = \prod_{j=1}^{m^*} \frac{Pr[\widehat{\beta}_i[j] = s_j | s_1, ...s_{j-1}]}{Pr[\widehat{\beta}_i'[j] = s_j | s_1, ...s_{j-1}]}$$

Each item of the product has the first $j-1$ tuples of the $\widehat{\beta}_i$-series fixed. Each s_j is the result of the Laplace noise that has been calibrated for β_i based on its sensitivity, as calculated using in Lemma 2. The sensitivity of this function is derived under the assumption that the two graphs have, at most, one edge difference. Thus, the conditional probabilities are Laplacians, which allows us to derive the following inequalities:

$$\prod_{j=1}^{m^*} \frac{Pr[\widehat{\beta}_i[j] = s_j | s_1, ...s_{j-1}]}{Pr[\widehat{\beta}'_i[j] = s_j | s_1, ...s_{j-1}]} \leq \prod_{j=1}^{m^*} e^{\frac{|\widehat{\beta}_i[j] - \widehat{\beta}'_i[j]|}{\sigma}}$$

By definition $\widehat{\beta}_i = \beta_i + Lap(\frac{S_{\beta_i}}{\epsilon})^{|\beta_i|}$ and by Lemma 2 $||\beta_i - \beta'_i||_1 \leq S_{\beta_i}$ with $S_{\beta_i} \leq 4d_{k_i} + 1$. Let σ_i be the scale parameter of the Laplacian noise applied in each cluster i, thus:

$$\prod_{j=1}^{m^*} e^{\frac{|\widehat{\beta}_i[j] - \widehat{\beta}'_i[j]|}{\sigma}} = e^{\frac{||\widehat{\beta}_i - \widehat{\beta}'_i||_1}{\sigma}}$$

$$= e^{\frac{||\widehat{\beta}_i + Lap(\frac{S_{\beta_i}}{\epsilon}) - \widehat{\beta}'_i - Lap(\frac{S_{\beta_i}}{\epsilon})||_1}{\sigma}} = e^{\frac{||\beta_i - \beta'_i||_1}{\sigma}} \leq e^{\frac{4d_{m_i}+1}{\frac{4d_{m_i}+1}{\epsilon}}}$$

Finally, by applying the logarithmic function the theorem statement is proved. □

Theorem 2 shows that adding noise does achieve provable ϵ-differential privacy on each cluster. In particular, we prove that by only leveraging m^* independent Laplace random variables, with parameter $\lambda = (\frac{S_{\beta_i}}{\epsilon})$, it is possible to generate sufficient noise per cluster to satisfy the privacy requirement.

Conquering ϵ-privacy into $\cup_i \widehat{\beta}_i$. Our next task is to leverage the proved ϵ-differential privacy of each independent $\widehat{\beta}_i$ to guarantee privacy on the entire perturbed $\widehat{\beta}$-series$= \cup_i \widehat{\beta}_i$. In order to achieve this goal a further step is required, shown in the following corollary.

Corollary 1. *The amount of information an attacker can learn on $\widehat{\beta}_i$ by observing any $\widehat{\beta}_j$ with $i \neq j$ is null.*

This proof considers only two sub-series for simplicity. Given Lemma 3, this proof can be extended to any number of clusters.

Proof. Let A and B be two sub-series built out of our partition strategy and let $\widehat{A}$ and $\widehat{B}$ be their ϵ-differentially private projection as proved in Theorem 2. Finally, let a and b be events on $\widehat{A}$ and $\widehat{B}$, respectively. Through the Shannon Entropy Theory we quantify the information a sub-series could exploit on another sub-series. In particular, the Mutual Information

$$I(\widehat{A};\widehat{B}) = \sum_{a,b} p(a,b) \log \frac{p(a,b)}{p(a)p(b)}$$

is the amount of information an attacker can infer on $\widehat{A}$ by observing $\widehat{B}$. By construction the sensitivity of the sub-series A is independent from the sensitivity of the sub-series B, as proved in Lemma 3. This means that the sub-series A is perturbed by a Laplace random process with parameter λ_A that is independent from the Laplace random process acting on B, as consequence of Lemma 2. Thus, this independence property directly implies that the Mutual Information $I(\widehat{A},\widehat{B}) = 0$, that is, an attacker gains no information on $\widehat{A}$ by observing $\widehat{B}$, which concludes the proof. □

The properties derived on the different β_is are sufficient to begin the **conquer phase** of our DRC approach. The goal of the conquer phase is to unify the $\widehat{\beta}_i$s such that the union set inherits the ϵ-privacy guarantees from the individual sub-series.

Theorem 3. *Given $\widetilde{m}$ different sub-series $\widehat{\beta}_i$ with $i = 1, ..., \widetilde{m}$, the result of the DRC conquer strategy $\cup_i \beta_i$ satisfies the ϵ-differential privacy property.*

Proof. The DRC strategy produces $\widetilde{m}$ ϵ-differentially private sub-series $\widehat{\beta}_i$, as proved in Theorem 2. Each β_i satisfies Lemma 2 and Lemma 3, and any com-

bination of $\widehat{\beta}_i$s satisfies Corollary 1. The privacy independence property, from Corollary 1, implies that $\cup_i \widehat{\beta}_i$ satisfies the ϵ-Differential Privacy property. □

Thus, we have proven that our perturbed dK-2, $\cup_i \widehat{\beta}_i$, satisfies the ϵ-differential privacy requirement. DRC achieves a tighter bound on noise than dk-PA due to the properties from Lemmas 2 and 3.

Error Analysis. We now quantify the error introduced to dK-2 via our DRC strategy. Error analysis on DRC is complicated because our algorithm does not specify the number of clusters to generate during partitioning. Instead, our clustering approach is general, and covers any possible set of cuts on the β-series such that the resulting sub-series differ in cardinality and sensitivity from each other, so long as they respect Lemmas 2 and 3. Therefore, in order to provide an error analysis that covers any possible clustering of the β-series we have to study both the lower and the upper bound of the error injected into those series.

Definition 4. *The error estimation of the union of the $\widehat{\beta}_i$s under the ∂ ordering on dK-2 of a graph G can be computed as the expected randomization in generating $\widehat{\beta} = \cup_i \widehat{\beta}_i$.*

The expected randomization in $\widehat{\beta}$ is quantified as

$$\sum_{i=1}^{\widetilde{m}} E\left(\sum_j (\widehat{\beta}_i[j] - \beta_i[j])^2\right) = \sum_{i=1}^{\widetilde{m}} |\beta_i| E[Lap(\frac{S_{\beta_i}}{\epsilon})^2]$$

The lower bound is found when each S_{β_i} have the same minimum value, which is 1, and thus

$$\sum_{i=1}^{\widetilde{m}} |\beta_i| E[Lap(\frac{S_{\beta_i}}{\epsilon})^2] \geq d_{max}^2 Var(Lap(\frac{1}{\epsilon})) = \Omega(\frac{d_{max}^2}{\epsilon^2})$$

Note that the considered minimum, *i.e.* 1, happens only when a graph of nodes with zero degree is considered, and after adding an edge S_β is 1. The upper bound is found when each S_{β_i} have the maximum value that, as proved in Lemma 2, is $O(d_{max})$, and thus

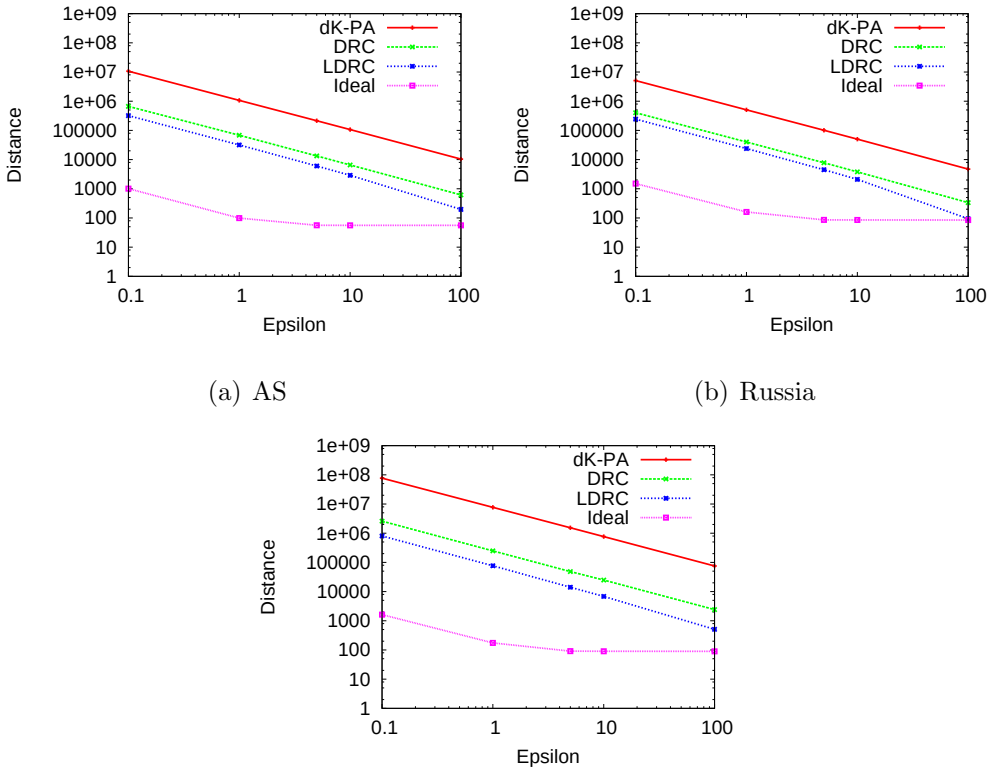
$$\sum_{i=1}^{\widetilde{m}} |\beta_i| E[Lap(\frac{S_{\beta_i}}{\epsilon})^2] \leq d_{max}^2 Var(Lap(\frac{d_{max}}{\epsilon})) = O(\frac{d_{max}^4}{\epsilon^2})$$

The worst-case error level of DRC is equal to that of dK-PA. However, depending on graph structure, the error level can decrease down to $\Omega(\frac{d_{max}^2}{\epsilon^2})$. As we demonstrate in the next section, real graphs exhibit error rates towards the lower bound. Thus, in practice, DRC performs much better than dK-PA.

4.4.3 Evaluating and Optimizing DRC

To quantify the improvement DRC achieves over the dK-PA strategy, we compare the results of applying each algorithm on our graphs. As before in Section 4.3.3, we quantify error using the Euclidean distances between each of their dK-2-series and the dK-2-series of the original graph. As seen in Figure 4.4, DRC reduces the Euclidean distance by one order of magnitude for different graphs and a range of ϵ values. As is the case for dK-PA, error introduced by DRC decreases exponentially as the value of ϵ increases, which is clear from the linear correlation in the log-log scale plot of Figure 4.4.

Further Optimization with LDRC. Despite its improvement over dK-PA, DRC is still quite far from the idealized function in terms of error (see Figure 4.4).



(a) AS

(b) Russia

(c) WWW

Figure 4.4: Euclidean distances of the dK-2-series of different ϵ-Differential Privacy strategies on three real graphs.

We apply a prior result from [72] that proves how to use isotonic regression [19], *i.e.* evenly "smooth" out the introduced noise across tuples, without breaking differential privacy properties. This technique enables a reduction of the error introduced in the dK-2-series by another constant factor.

Formally, given a vector p of length p^*, the goal is to determine a new vector p' of the same length which minimizes the L_2 norm, *i.e.* $||p-p'||_2$. The minimization problem has the following constraints: $p'[i] \leq p'[i+1]$ for $1 \leq i < p^*$. Let $p[i,j]$

be a sub-vector of length $j - i + 1$, that is: $< p[i], ..., p[j] >$. Let define $M[i, j]$ as the mean of this sub-vector, *i.e.* $M[i, j] = \sum_{k=i}^{j} p[k]/(j - i + 1)$.

Theorem 4. *[19] The minimum L_2 vector, p', is unique and is equal to $p'[k] = \widetilde{M_k}$, with:*

$$\widetilde{M_k} = min_{j \in [k,p^*]} max_{i \in [1,j]} M[i, j]$$

We apply this technique on the set of all tuples produced by DRC. We refer to it as the L_2 minimization Divide Randomize and Conquer algorithm, or LDRC. We include LDRC in our comparison of algorithms in Figure 4.4, and see that LDRC provides roughly another 50% reduction in error over the DRC algorithm. Since it consistently outperforms our other algorithms, we use LDRC as the algorithm inside the Pygmalion graph model.

Implications. Finally, we note that our DRC partition technique is general, and has potential implications in other contexts where it is desirable to achieve differential privacy with lower levels of injected noise. More specifically, it can serve to reduce the amount of perturbation necessary when the required perturbation is a function of a parameter that varies significantly across values in the dataset.

4.5 End-to-end Graph Similarity

We have already quantified the level of similarity between real and synthetic graphs by computing the Euclidean distances between their respective dK-series datasets. These values represent the distortion in the statistical representation of a graph, *i.e.* the dK-series, but do not capture the ultimate impact of the

added noise on graph structure. In this section, we evaluate how well Pygmalion preserves a graph's structural properties by comparing Pygmalion's differentially private synthetic graphs against the originals in terms of both graph metrics and outcomes in application-level tests. Strong structural similarity in these results would establish the feasibility of using these differentially private synthetic graphs in real research analysis and experiments.

4.5.1 Graph Metrics

Our evaluation includes two classes of graph metrics. One group includes *degree-based metrics* such as: Average Node Degree, Degree Distribution, Joint Degree Distribution and Assortativity. These are basic topological metrics that characterize how degrees are distributed among nodes and how nodes with particular degree connect with each other. The second group includes *node separation metrics* that quantify the interconnectivity and density of the overall graph. This group includes metrics such as Graph Diameter, Radius and Average Path Length.

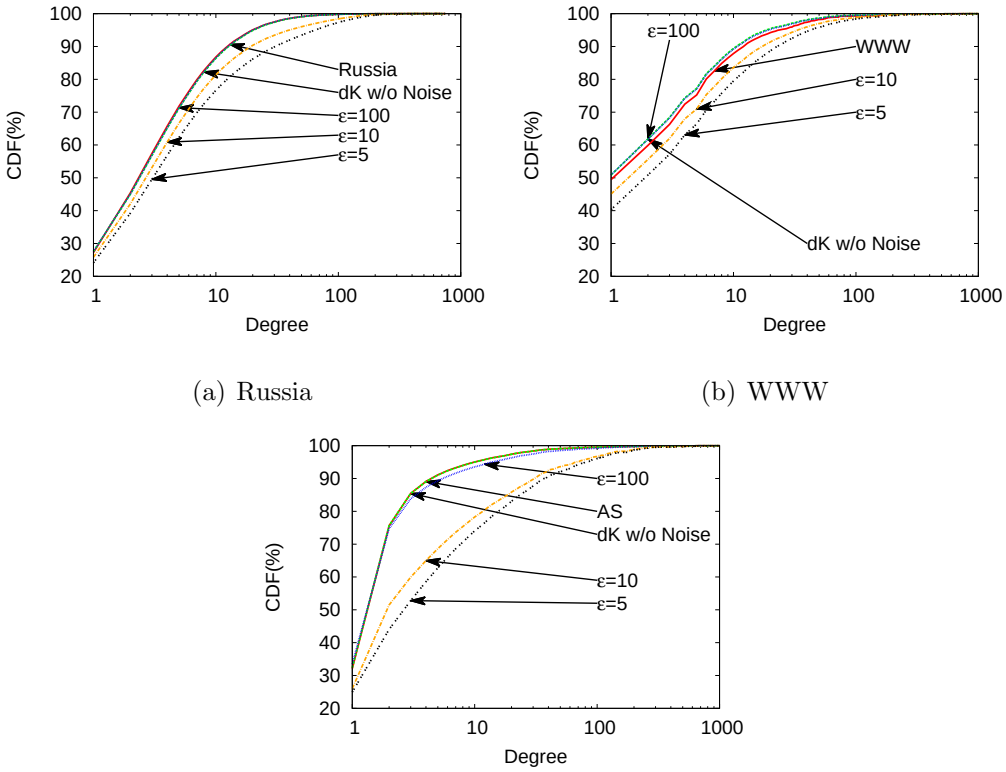
For our evaluation purposes, we always use our most advanced algorithm, *i.e.* Pygmalion LDRC. We only focus on Pygmalion LDRC, because there are practical problems in generating large graphs from dK values after significant noise has been added. As shown earlier, the dK-PA model introduces the highest noise. In fact, errors introduced by dK-PA are so large that the generator fails when trying to generate large graphs with the resulting noisy dK distributions.

We generate ϵ-private graphs for $\epsilon \in [5, 100]$, and compare the graph metrics of the resulting synthetic graphs against those of the original graph, and a synthetic graph generated by the dK model with no additional noise added. We limit

ourselves to ϵ-private graphs with $\epsilon \in [5, 100]$ because of three reasons. First, we aim to find the ϵ value that contributes to a smallest noise such that it is statistically similar to the synthetic dK-2 graph with no privacy enforced. This way, we can indirectly quantify the level of privacy introduced by a pure synthetic graph with no additional steps taken to improve privacy. This by itself is a potentially interesting result. In particular, we obtain this property only when ϵ is equal to 100. Second, the dK-2 distribution is a very sensitive function and it naturally requires a high level of noise to provide strong levels of privacy guarantees. Unfortunately, very small values of ϵ require larger noise values, thus producing synthetic graphs that are extremely different in structure from the original. Finally, for $\epsilon < 1$, the required noise level is so high for larger graphs, that the dK graph generator fails to produce synthetic graphs that match the resulting dK distributions. This is clearly a limitation of the current system, one that we hope will be removed with the discovery of less sensitive models and optimization techniques to further reduce noise required for ϵ-differential privacy.

As we mentioned, our results are highly consistent across our pool of graphs (Table 4.1), and we only report experimental results on three graphs: the Russia Facebook graph, the AS graph and the WWW graph.

Degree-based Metrics. These metrics are fundamental in understanding the statistical properties of node degrees and how nodes connect to each other to form specific topological structures. Out of the four metrics mentioned above, we report results for Degree-Distribution (which supersedes average node degree) and Assortativity (which is related to joint degree distribution).



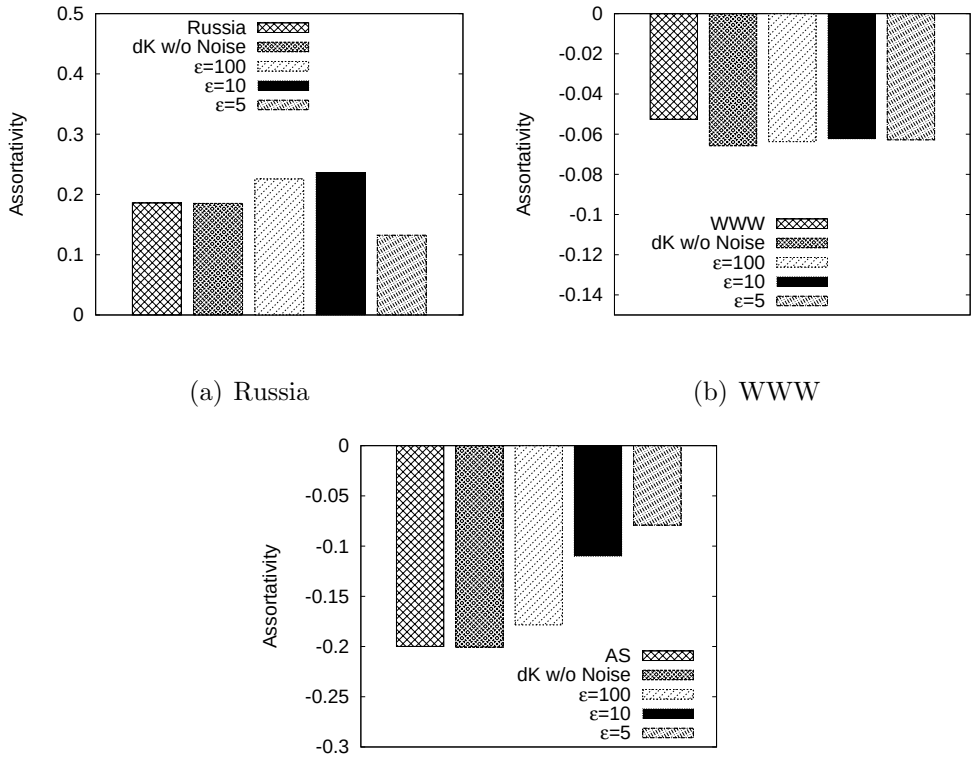
(a) Russia

(b) WWW

(c) AS

Figure 4.5: Degree distribution of three real measured graphs, *i.e.* Russia, WWW and AS, each compared to the dK-synthetic graph without noise and Pygmalion synthetic graphs with different ϵ values.

Degree Distributions. Figure 4.5 compares the node degree CDFs. For each of the Russia, WWW, and AS graphs, the degree distributions of both the Pygmalion (ϵ=100) graph and the dK-synthetic graph very closely match the degree distribution of the original graphs. When we increase the strength of the privacy guarantees, *i.e.* smaller ϵ values of 5 and 10, the accuracy of the synthetic degree distribution progressively decreases. For example, both the Russia and WWW graphs show a small deviation from the original distribution even for $\epsilon = 5$. Across



(a) Russia

(b) WWW

(c) AS

Figure 4.6: Assortativity of three real measured graphs, *i.e.* Russia, WWW and AS, each compared to the dK-synthetic graph without noise and Pygmalion synthetic graphs with different ϵ values.

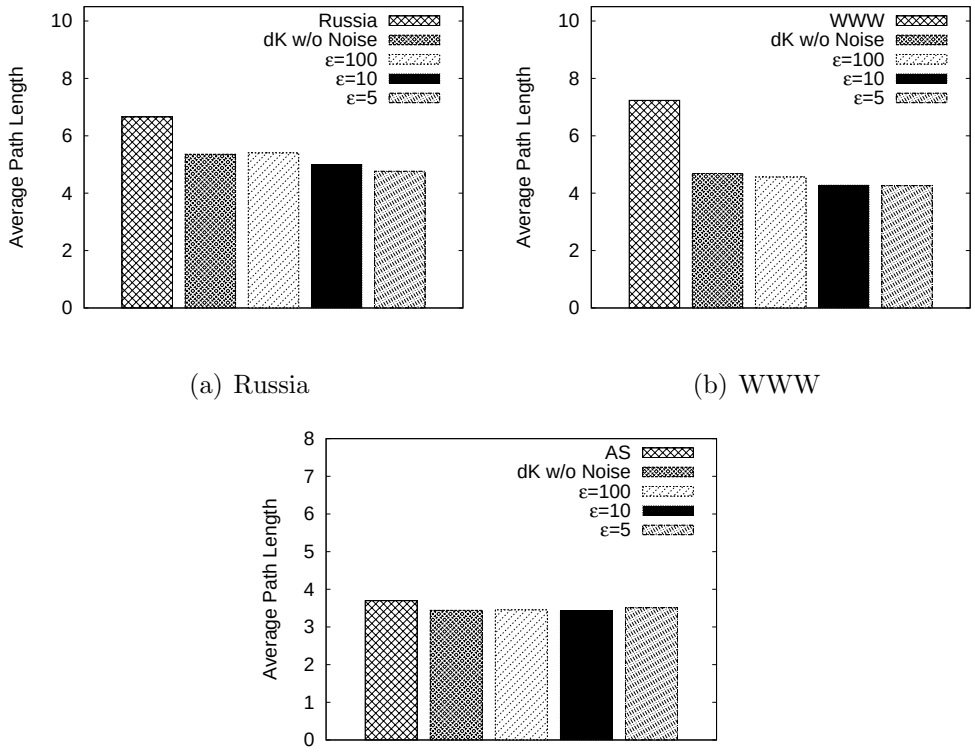
all models for these two graphs, the worst-case degree distribution deviation is still within 10% of the original.

The AS graph, on the other hand, shows a slightly different behavior. For small ϵ values, *i.e.* $\epsilon = 5$ and $\epsilon = 10$, the largest error is within 35% from the original graph values. The AS graph shows a different behavior because a small number of high degree nodes connect the majority of other nodes. Thus, when the privacy perturbation hits those high-degree nodes, it can produce structural changes that send ripples through the rest of the graph.

Assortativity. Figure 4.6 reports the results of the assortative metric computed on both real and synthetic graphs for each of the three graphs (Russia, WWW and AS). The assortativity metric describes the degree with which nodes with similar degree are connected to each other. Positive assortativity value denotes a positive correlation between the degrees of connected nodes, and negative values indicate anti-correlation. Note that both the WWW and AS graphs show negative assortativity (Figure 4.6(b) and Figure 4.6(c)).

As with the degree distribution results, for each of our graphs (Russia, WWW, and AS), assortativity results from synthetic graphs for $\epsilon = 100$ and those from the dK-series closely match results from the original graphs. As we increase the level of privacy protection, the results get slightly further from the original values. For example, using $\epsilon = 5$ on Russia produces an error less than 0.05 on the assortativity value. The same ϵ value for the WWW graph produces negligible error on assortativity. Assortativity results on the AS graph are also consistent with degree distribution results. Under high privacy requirements, *i.e.* $\epsilon = 5$, error on assortativity reaches 0.12.

Node Separation Metrics. For brevity, we report only the Average Path Length as a representative of the node separation metrics. Figure 4.7 shows the Average Path Length (APL) values computed on Russia, WWW and AS compared to the APL values on their synthetic graphs. On Russia and WWW, APL results denote a moderate level of error (higher when compared to results for the earlier graph metrics). We can see that the error is mainly introduced by the impreciseness of the dK-model, since the synthetic graph from the dK-series with no noise shows the same error. In comparison, the error introduced by strengthening



(a) Russia

(b) WWW

(c) AS

Figure 4.7: Average path length of three real measured graphs, *i.e.* Russia, WWW and AS, each compared to the dK-synthetic graph without noise and Pygmalion synthetic graphs with different ϵ values.

privacy (and hence decreasing ϵ) is relatively small. This is encouraging, because we can eliminate the bulk of the error by moving from dK-2 to a more accurate model, *e.g.* dK-3.

As with previous experiments, the AS graph shows a slightly different behavior. In this case, all of our synthetic graphs do a good job of reproducing the average path length value of the AS graph.

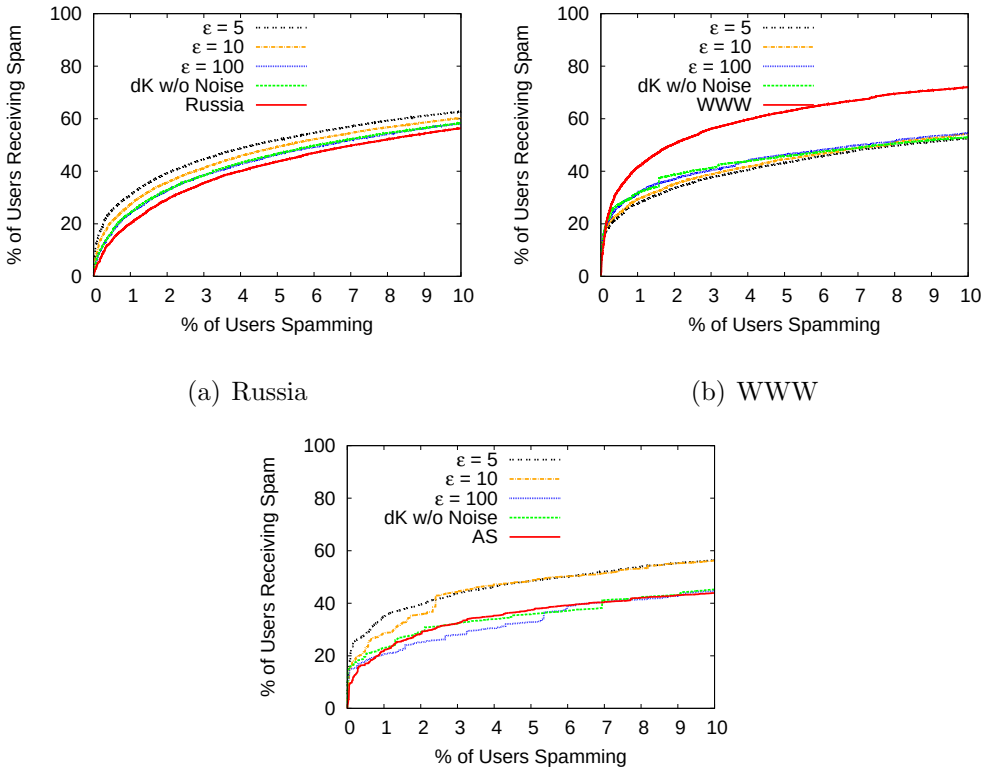
Summary. Our experimental analysis shows that synthetic graphs generated by Pygmalion exhibit structural features that provide a good match to those of the original graphs. As expected, increasing the strength of privacy guarantees introduces more noise into the structure of the synthetic graphs, producing graph metrics with higher deviation from the original graphs. These observations are consistent across social, web, and Internet topology graphs.

Overall, these results are very encouraging. They show that we are able to effectively navigate the tradeoff between accuracy and privacy by carefully calibrating the ϵ values. The fact that significant changes in ϵ values do not dramatically change the graph structure means owners of datasets can guarantee reasonable levels of privacy protection and still distribute meaningful graphs that match the original graphs in structure.

4.5.2 Application Results

For a synthetic graph to be usable in research, ultimately it must produce the same results in application-level experiments as the original graph it is replacing. To quantify the end-to-end impact of trading graph similarity for privacy protection, we compare the results of running two real world applications on both differentially private synthetic graphs and the original graphs. We implement two applications that are highly dependent on graph structure: Reliable Email (RE) [60] and Influence Maximization [33].

Reliable Email. RE [60] is an email spam filter that relies on a user's social network to filter and block spam. One way to evaluate the security of RE is to compute the number of users in a network who can be spammed by a fixed



(a) Russia

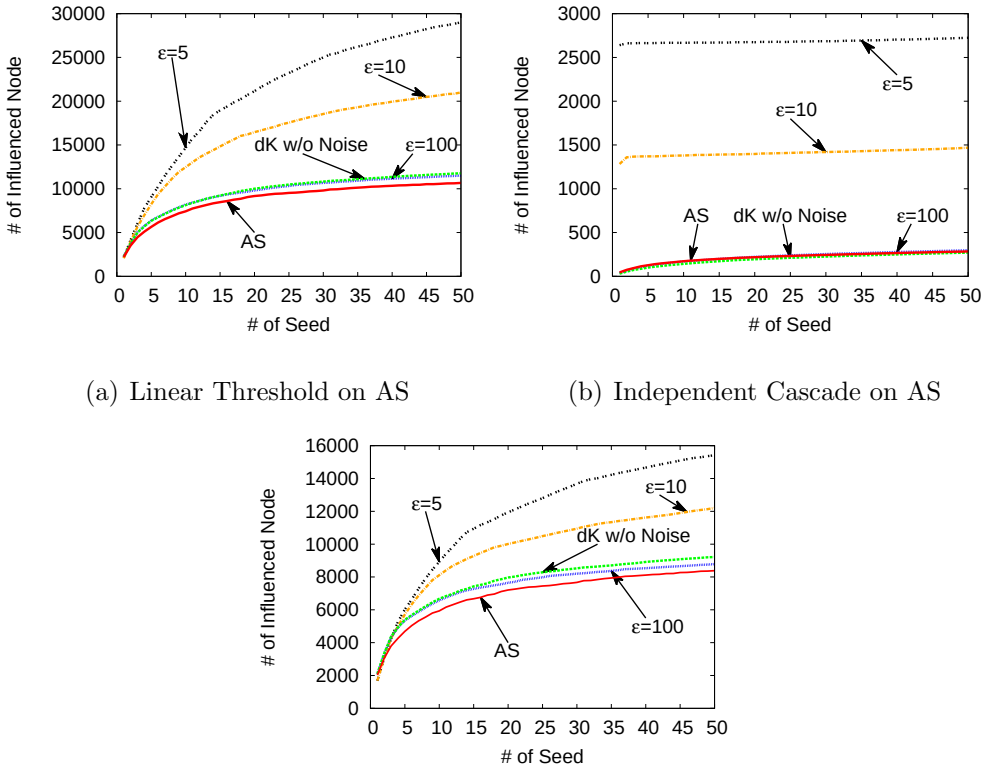
(b) WWW

(c) AS

Figure 4.8: Reliable Email (RE) experiment run on three real measured graphs, *i.e.* Russia, WWW and AS, each compared with the dK-synthetic graph without noise and Pygmalion synthetic graphs with different ϵ values.

number of compromised friends in the social network. This experiment depends on the structure of the network, and is a useful way to evaluate whether Pygmalion graphs can be true substitutes for measurement graphs in research experiments.

Figure 4.8 shows the portion of the nodes flooded with spam as we increase the number of malicious spammers, using different graphs as the underlying social network topology. We show results on the usual three graphs, Russia, WWW and AS. On the Russia Facebook graph, all synthetic graphs closely follow the original graph. Even in the case of the strongest privacy setting, *i.e.* $\epsilon = 5$, the difference



(a) Linear Threshold on AS

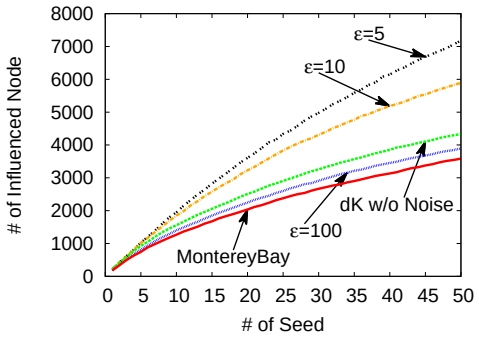
(b) Independent Cascade on AS

(c) Weighted Cascade on AS

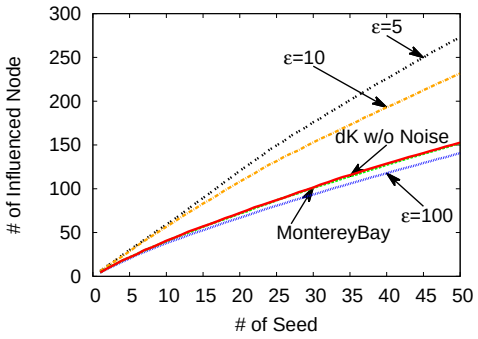
Figure 4.9: Results of the Degree Discount Influence Maximization algorithm on the AS graph, compared to dK graphs without added noise, and Pygmalion synthetic graphs with different ϵ values.

between the synthetic graph result and those of the original is at most 10%. For both the WWW and AS graphs, all synthetic graphs with and without noise produce results within 20% of the original graphs.

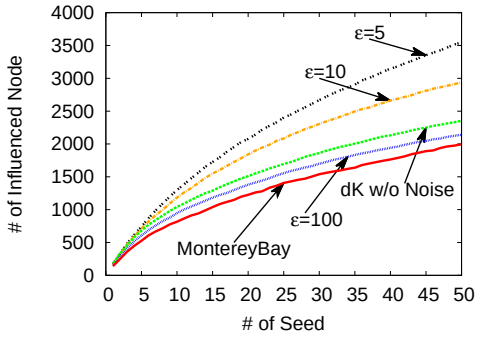
Influence Maximization. The influence maximization problem tries to locate users in the network who can most quickly spread information through the network. This problem is most commonly associated with advertisements and public relations campaigns. Evaluating a solution to this problem includes two steps.



(a) Linear Threshold on MontereyBay



(b) Independent Cascade on MontereyBay



(c) Weighted Cascade on MontereyBay

Figure 4.10: Results of the Degree Discount Influence Maximization algorithm on the MontereyBay graph.

First, the solution must identify the nodes who can maximize influence in the network. Second, it must model the spread of influence through the network to quantify how many users the influence has ultimately reached.

For our purposes, we use a recently proposed heuristic for influence maximization that minimizes computation. The heuristic is called the Degree Discount method [33], and is able to find the most influential nodes, called "seeds," on a given graph. Starting from those seed nodes, we run three different influence dissemination models: Linear threshold (LT), Independent Cascade (IC) and

Weighted Cascade (WC), to determine the total number of users in the network influenced by the campaign. We use source code we obtained from the authors. However, significant memory overhead in the code meant that we had to limit our experiments to smaller graphs. Therefore, we use the MontereyBay Facebook graph and the AS network topology graph in this experiment.

For both AS and MontereyBay graphs and each of the three influence dissemination models, Figure 4.9 and 4.10 shows the expected number of influenced nodes when increasing the number of initial seed nodes. While the actual percentage of users influenced varies across dissemination models, there are clear and visible trends. Results on the AS graph in Figures 4.9(a), 4.9(b), 4.9(c) all show that Pygmalion with $\epsilon = 100$ and the dK-synthetic graph without noise are almost identical to the original AS graph under all three dissemination models. Graphs with stronger protection, Pygmalion $\epsilon = 10$ and $\epsilon = 5$, progressively diverge from the results of the AS graph. Results on the MontereyBay graph are shown in Figures 4.10(a), 4.10(b), 4.10(c), and are quite similar to those on the AS graph. They confirm that Pygmalion $\epsilon = 100$ produces near perfect results, but higher privacy protection increases the deviations from results on the original MontereyBay graph.

4.5.3 Summary of Evaluation

We have used both popular graph metrics and application-level tests to evaluate the feasibility of using differentially private synthetic graphs in research. Our tests are not comprehensive, and cannot capture all graph metrics or application-level experiments. However, they are instructive because they show the observable

impact on graph structure and research results when we replace real graphs with differentially private Pygmalion graphs.

Our results consistently show that Pygmalion introduces limited impact as a result of adding noise to guarantee privacy. In fact, many of the largest errors can be attributed to limitations of the dK-2 series. Given the significant demand for realistic graphs in the research community, we expect that generator algorithms for more complex dK models will be discovered soon. Moving to those models, *e.g.* dK-3, will eliminate a significant source of error in these results.

4.6 Summary

We study the problem of developing a flexible graph privacy mechanism that preserves graph structures while providing user-specified levels of privacy guarantees. We introduce *Pygmalion*, a differentially-private graph model that aims these goals using the dK-series as a graph transformation function. First, we use analysis to show that this function has a high sensitivity, *i.e.* applied naively, it requires addition of high levels of noise to obtain privacy guarantees. We confirm this on both social and Internet graphs. Second, we develop and prove a partitioned privacy technique where differential privacy is achieved as a whole when it is achieved in each data cluster. This effectively reduces the level of noise necessary to attain a given level of privacy.

We evaluate our model on numerous graphs that range in size from 14K nodes to 1.7 million nodes. Our partitioned privacy technique reduces the required noise by an order of magnitude. For moderate to weak levels of privacy guarantees,

the resulting synthetic graphs closely match the original graphs in both graph structure and behavior under application-level experiments.

We believe our results represent a promising first step towards enabling open access to realistic graphs with privacy guarantees. The accuracy of our current model is fundamentally limited by both the degree of descriptiveness of dK-2 series, and the high noise necessary to inject privacy properties. There are two ways to improve our results. One way is to use a more descriptive, higher-order dK model, under the assumption that its sensitivity is reasonable low. While generators for higher order dK-models are still unknown, our techniques are general, and can be applied to obtain more accurate models as higher-order dK generators are discovered. Another way to improve is to discover a function (or model) of graph structure with much lower sensitivity. If such a function exists, it can potentially lower the noise required for a given privacy level by orders of magnitude.

Chapter 5

Conclusion

In this section, we first summarize our work on analyzing and processing big real graphs. We then share our lessons and wisdom learned from our work. Hopefully, this provides useful guidelines for researchers working in this area. Finally, we discuss future directions that we will pursue.

5.1 Summary

A growing number of big real graphs become available due to the recent explosive growth of networks. They are significantly different from the graphs in prior studies in terms of scale, level of dynamics, and structure. New challenges emerge in studying these big real graphs. In this dissertation, we focus on three key problems in analyzing and processing big real graphs, including node distance computation, dynamic graph analysis and modeling, and graph privacy. For each problem, we propose novel solutions, and evaluate the performance on a range of big real graphs.

First, to efficiently compute node distances, such as shortest path distances and random walk distances, we propose graph coordinate systems. To accurately

embed shortest path distances on massive graphs, we implement a hyperbolic graph coordinate system with naturally parallel embedding process. We also study the possibility to embed random walk distances using graph coordinate systems. Since traditional geometric spaces cannot capture the asymmetry of random walks, we design a novel space with two independent directional heights to account for the asymmetry. Through our extensive experiments on graphs from various networks, we show that using graph coordinate systems, node distances can be accurately estimated in microseconds, which can support real-time applications.

Next, we study dynamics in a large online social network. Given the first two-year growth of the Renren network, we first analyze its structural evolution at multiple network scales. Through the measurement, we observe users' activities are significantly impacted by the processes at different network scales, and identify several evolutionary properties in Renren network structure. We then explore our efforts to detect self-similar properties in Renren edge creation process. Using three popular measurement methods, we not only reliably identify self-similarity in the edge creation process, but also detect the time scale over which the self-similar property exists. Based on these observations, we propose a new dynamic model including a temporal component and a structural component. Using the Renren dynamic dataset to fit this model, the graphs generated by the model reproduce the sequence of edge creation events in absolute time, which exhibits the observed dynamic properties.

Last, we tackle privacy issue in sharing graph datasets. Observing the tension between graph structure utility and graph privacy, we develop a differentially-private graph model. In the basic design, we directly add a controlled level of noise into the dK-2 series, and use the perturbed dK-2 series to generate syn-

thetic graphs. Although this design can achieve the required level of privacy, our theoretical and empirical analysis on big real graphs shows that the amount of noise grows polynomially with the maximum degree of a graph. In other words, high distortion can be introduced into graph structure. To improve the accuracy of the generated graphs, we develop a Divide Randomize and Conquer algorithm, and prove this algorithm not only maintains the same level of privacy but also reduces the noise required by differential privacy. Our end-to-end experiments on a range of big real graphs confirm that the synthetic graphs generated by the improved model are similar to the original graphs in terms of graph metrics and application-level performance.

5.2 Lessons

Through studying the problems, we have learned three important lessons in analyzing and processing big real graphs. We summarize them as follows, hoping to provide high-level guidance for researchers working in this direction.

Scale with Big Graph Size via. Approximation. Scalability becomes one of the most important problems in analyzing and processing big real graphs. Many efforts have been made to address this challenge, such as parallel systems, new algorithms with lower bound, and approximation methods. Among them, approximation methods are one promising direction to efficiently process big real graphs.

We learned this lesson from our work on node distance computation (Chapter 2). Given a graph with millions of nodes and billions of edges, the time to

compute the distance between one pair of nodes can take up to hours. This is a big obstacle for both graph analysis and real-time applications using node distances. To efficiently compute node distances, we propose an approximation approach by embedding graphs into a geometric space. With one-time precomputation, node distances can be accurately estimated in constant time (microseconds). This result is important for large graph analysis, and has significant meaning for practical applications.

Although it is not panacea, approximation methods raise the hope to scalably process large graphs. In big real graphs, it is often difficult to obtain exact results for many problems, such as modularity, betweenness centrality, and subgraph matching. With approximation methods, we may efficiently compute an accurate result, which is meaningful in understanding graph structure and helpful to support related applications.

Balance Tradeoff Based on Realistic Needs. Many system or algorithm designs on top of big real graphs face the challenge: how to prioritize multiple goals. For example, in node distance computation, accuracy and efficiency are two important goals. Each of them can be achieved by sacrificing the performance of the other one. We find that efficiency is more important in many practical applications, such as distance based search and friend recommendation. Therefore, our design emphasizes the goal of efficiency, and proposes a constant-time distance estimation method with small errors. This instance demonstrates the importance that when designing graph systems with multiple goals that may not be easily attained at the same time, as system designers, we need to understand the real needs so as to better balance the tradeoff between goals.

Another example on navigating tradeoff between goals is our work on graph privacy in Chapter 4. Strong privacy guarantee and high utility of graph structure are two extremes in the study space. In prior studies, researchers tend to bias in favor of one extreme. For example, k-degree anonymization [111] maintains low distortion in graph structure but only protects graph privacy against a specific attack. However, from our observations, both graph privacy and structure utility are important for real world applications. Thus, in our design, we make a reasonable adjustment between them using differential privacy.

Although it is not easy to identify the needs sometimes, it is important for designing practical graph systems. To make proper tradeoff between design goals, one possible way to uncover realistic needs is to explore the demands from network operators and graph application developers.

Invalid Traditional Graph Assumptions. Small graph analysis in prior work helped to build some useful graph assumptions or models, such as preferential attachment. Because of fast growth of different networks, some of the traditional assumptions may not be applicable in modeling the graphs abstracted from these networks.

For example, we find the decay of the strength of preferential attachment in Renren. In particular, in the preferential attachment model, new nodes select destination nodes with probability proportional to nodes' degree. That means, the strength of preferential attachment is constant. But in Section 3.3.1, we show as network grows, the strength weakens. This result does not say that the preferential attachment model is false. It is just because there was no chance to validate its effectiveness on large, high dynamic real graphs. From this example,

we learn that due to the significant difference of big real graphs, it is necessary to validate traditional graph assumptions before using them in big real graph study.

5.3 Future Work

Our work in this dissertation introduces novel solutions to three important problems in studying big real graphs. Because of the fast evolution of networks, our solutions need to adapt to such dramatic changes in big real graphs. In this section, we discuss three potential directions in analyzing and processing big real graphs.

5.3.1 Processing and Querying Large Dynamic Graphs

Today's complex networks are highly dynamic. The graphs from these networks dramatically grow and change over time. While they continue to fast evolve, few work has worked on processing and querying these large dynamic graphs. Here we target two of fundamental computational problems in dynamic graphs: node distances computation in dynamic graphs, and link prediction.

Embedding Dynamic Graphs. Node distance is difficult to compute on big real graphs. As complex networks continue to thrive with fast pace, fast node distance computation in dynamics graphs is of significant impact on understanding and processing the graphs from these highly dynamic networks. While most of the proposed solutions focus on computing node distance in static graphs, few work is reported in dealing with this problem in dynamic graphs, especially at large scale. Our goal is to fast reveal the changes of node distances in dynamic graphs.

Inspired by our work on graph coordinate systems, we consider to implement a dynamic graph embedding system.

To embed dynamic graphs, we face two key challenges. First, it is crucial to fast update the distances between landmarks and all other nodes, referred to as ground truth distances. Recall that as the essential of graph coordinate systems, ground truth distances are used to calibrate node positions in a geometric space. Thus, in dynamic graphs, it is important to efficiently renew the ground truth distances. Recently, several algorithms have been proposed to incrementally update shortest path in dynamic graphs [97, 4]. We may adopt one of such algorithm to solve this problem. Second, once any of ground truth distances changes, we encounter the problem: how to efficiently recompute graph coordinates. Although we can parallel the embedding process to reduce the embedding time, it is still expensive to compute the coordinates for all the nodes in a large graph.

Link Prediction. In complex networks, especially online social networks, large numbers of new edges arrives everyday, which signify the appearance of new interactions between nodes. Predicting edge creations is one fundamental problem for dynamic graph study. Many solutions have been proposed to address this problem [107, 15, 76, 108]. These studies focused on improve the accuracy of prediction, whereas few of them are validated on real dynamic graphs at massive scale. Moreover, little work is known about how various link prediction algorithms work in the context of large dynamic graphs. Thus, we aim to fill this research gap. We will use big real dynamic graphs, like Renren used in Chapter 3, to compare the performance of the various link prediction methods. Based on our analysis, we may develop a more accurate algorithm to predict link formation.

5.3.2 Applications on Graph Coordinate Systems

As shown in Chapter 2, graph coordinates systems can be used in node distance based applications, such as social-distance based search rank, graph separation metric computation, and link prediction. In the future, we will explore more applications, which graph coordinate systems help to reduce their computation complexity. Now we discuss two potential applications, including graph matching, and community detection.

Graph Matching. Graph matching is a fundamental problem in graph study and of practical importance. Graph matching, also known as graph isomorphism problem, is a one-to-one mapping between nodes in two graphs such that an edge exists between any two nodes in a graph if and only if their mapped nodes in the other graph has one edge. As a classical graph computational problem, graph matching is known for its high computation cost.

To solve this problem, we consider to use graph coordinate systems to efficiently determine whether two graph matches. Intuitively, if two isomorphism graphs use same landmarks, same nodes in both graphs should be embedded into the same space positions such that their coordinates are the same. Thus, given the coordinates of two graphs, graph matching problem is reduced to compare the node coordinates in the two graphs. However, simple pairwise comparison is still of high computation complexity. That is, to match two n-node graphs, it takes $O(n^2)$ pairs of coordinate comparison. Thus, our goal is to design an algorithm to efficiently compare node coordinates in large graphs.

Community Detection. Structural community is an important notion to cluster graph nodes, and widely used in graph analysis and applications. A structural community is defined as a group of nodes where more edges resides than outside. In the past, various metrics are proposed to quantity how well a graph is clustered into communities, such as modularity [130] and conductance [154], and different algorithms are developed to detect communities [130, 132, 131, 35, 172, 25, 106]. Most of them suffer the scalability problem, *i.e.* efficiently identify communities in large graphs. Intuitively, because of high density inside communities, community nodes are closer to each other than to nodes outside the communities. As an alternative, we can use random walk distances as a metric to define communities. That is, if a group of nodes are closer to each other in terms of random walk distances than to other nodes in the graph, we say these nodes form a community. Under this definition, we can apply embedded graph coordinates to identify communities. Similar to graph matching problem, instead of pairwise distance computation, we need to explore an efficient method to use graph coordinates to detect communities.

5.3.3 Graph Watermarking

Today's graphs represent sensitive information. Controlling access to these datasets is a difficult challenge. More specifically, instead of sharing sensitive datasets publicly, the data owners would like to share the data with trusted entities. For example, a large social network like Facebook may choose to share its social graph with trusted collaborators, but do not want to leak the data into the

broader research community. Even so, it is still challenging to prevent the shared datasets from being leaked.

To prevent data leakage, we propose a new solution, graph watermarks. Intuitively, graph watermarks are small, and serve to associate some metadata to the data, like the information of the data owner. Thus, once a shared dataset is found in later, the data owner can extract the watermark from the leaked copy, and use it as proof to seek damages against the collaborator responsible for the leak.

An effective graph watermark system needs to provide several key properties. First, graph watermarks should be small, which introduces small distortion on graph structure and cannot be easily detected by attackers. Second, watermarks should be unique, which is difficult to forge and should not occur naturally in graphs. Third, both embedding and extraction of watermarks should be efficient, even in extremely large graphs. The last and the most important, a watermark system works in any application involving graphs. Thus, we make no assumption about graphs. Instead, our system works on symmetric, unweighted graphs without any node or edge labels. To achieve the goals, we need to explore the possible designs to embed and extract watermarks in graphs, prove the uniqueness of the design watermarks, and evaluate the robustness of the watermarks.

Bibliography

[1] P. Abry and D. Veitch. Wavelet analysis of long-range-dependent traffic. *IEEE TOIT*, 44(1):2–15, 1998.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Approximation algorithms for k-anonymity. In *Proc. of ICDT*, 2005.

[3] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. Analysis of topological characteristics of huge online social networking services. In *Proc of WWW*, 2007.

[4] T. Akiba, Y. Iwata, and Y. Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proc of WWW*, 2014.

[5] L. Akoglu and C. Faloutsos. RTG: a recursive realistic graph generator using random typing. *Machine Learning and Knowledge Discovery in Databases*, pages 13–28, 2009.

[6] L. Akoglu, M. McGlohon, and C. Faloutsos. RTM: Laws and a recursive generator for weighted time-evolving graphs. In *Proc. of ICDM*, 2008.

[7] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the World-Wide Web. *Nature*, 401(6749):130–131, 1999.

[8] R. Albert, H. Jeong, and A.-L. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406:378–382, July 2000.

[9] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proc. of FOCS*, 2006.

[10] S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM TKDD*, 3(4):16, 2009.

[11] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. Technical report, SIAM J. Numer. Anal, 2005.

[12] L. Backstrom. Anatomy of facebook, Nov 2011. `https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859`.

[13] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proc of WWW*, 2007.

[14] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proc. of KDD*, 2006.

[15] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proc. of WSDM*, 2011.

[16] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *Proc. of SIGMOD*, 2011.

[17] Z. Bar-Yossef and L.-T. Mashiach. Local approximation of pagerank and reverse pagerank. In *Proc. of CIKM*, 2008.

[18] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509, 1999.

[19] R. E. Barlow and H. D. Brunk. The isotonic regression problem and its dual. *Journal of the American Statistical Association*, 67(337):140–147, 1972.

[20] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proc. of CRYPTO*, 2007.

[21] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user behavior in online social networks. In *Proc. of IMC*, 2009.

[22] J. Beran. *Statistics for long-memory processes*. Chapman & Hall/CRC, 1994.

[23] J. Beran, R. Sherman, M. S. Taqqu, and W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43(234):1566–1579, 1995.

[24] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *PVLDB*, 2(1):766–777, 2009.

[25] V. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *CoRR*, abs/0803.0476, 2008.

[26] A. Blum, T. Chan, and M. Rwebangira. A random-surfer web-graph model. In *Proc. of ANALCO*, 2006.

[27] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the sulq framework. In *Proc. of PODS*, 2005.

[28] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proc. of STOC*, 2008.

[29] A. Bonato, N. Hadi, P. Horn, P. Prałat, and C. Wang. A dynamic model for on-line social networks. *Algorithms and Models for the Web-Graph*, pages 127–142, 2009.

[30] F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V. S. Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[31] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proc. of SDM*, 2005.

[32] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proc. of WWW*, pages 571–580, 2007.

[33] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proc. of KDD*, 2009.

[34] H. Chun, H. Kwak, Y. Eom, Y. Ahn, S. Moon, and H. Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In *Proc. of IMC*, 2008.

[35] A. Clauset, M. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6), 2004.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3 edition, 2009.

[37] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical internet coordinates for distance estimation. In *Proc. of ICDCS*, 2004.

[38] D. Cox. Long-range dependence: A review. *Statistics: An Appraisal*, 1984.

[39] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM TON*, 5(6):835–846, 1997.

[40] A. Cvetkovski and M. Crovella. Hyperbolic embedding and routing for dynamic graphs. In *Proc. of INFOCOM*, 2009.

[41] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of SIGCOMM*, 2004.

[42] A. Das Sarma, S. Gollapudi, M. Najork, and R. Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proc. of WSDM*, 2010.

[43] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal. Fast distributed pagerank computation. In *Proc. of ICDCN*, 2013.

[44] I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Physical review letters*, 94, 2005.

[45] X. Dimitropoulos, D. Krioukov, A. Vahdat, and G. Riley. Graph annotations in modeling complex network topologies. *ACM Trans. Model. Comput. Simul.*, 19:17:1–17:29, 2009.

[46] S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. *Adv. Phys*, pages 1079–1187, 2002.

[47] C. Dwork. Differential privacy. In *Proc. of ICALP*, 2006.

[48] C. Dwork. Differential privacy: A survey of results. In *Proc. of TAMC*, 2008.

[49] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proc. of EUROCRYPT*, 2006.

[50] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. of IACR TCC*, 2006.

[51] Z. Eisler, I. Bartos, and J. Kertész. Fluctuation scaling in complex systems: Taylor's law and beyond 1. *Advances in Physics*, 57(1):89–142, 2008.

[52] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlós. Towards scaling fully personalized pageRank: algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005.

[53] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[54] P. Francis, S. Jamin, V. Paxon, L. Zhang, D. Gryniewicz, and Y. Jin. An architecture for a global internet host distance estimation service. In *Proc. of INFOCOM*, 1999.

[55] A. Friedman, R. Wolff, and A. Schuster. Providing k-anonymity in data mining. *The VLDB Journal*, 17(4):789–804, 2008.

[56] Y. Fu, Z. Chen, G. Koru, and A. Gangopadhyay. A privacy protection technique for publishing data mining models and research data. *ACM Trans. Manage. Inf. Syst.*, 1:7:1–7:20, 2010.

[57] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao. Detecting and characterizing social spam campaigns. In *Proc. of IMC*, 2010.

[58] S. Garg, T. Gupta, N. Carlsson, and A. Mahanti. Evolution of an online social aggregation network: an empirical study. In *Proc. of IMC*, 2009.

[59] M. W. Garrett and W. Willinger. Analysis, modeling and generation of self-similar vbr video traffic. In *Proc. of SIGCOMM*, 1994.

[60] S. Garriss, M. Kaminsky, M. Freedman, B. Karp, D. Mazières, and H. Yu. Re: reliable email. In *Proc. of NSDI*, 2006.

[61] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A case study of unbiased sampling of OSNs. In *Proc. of INFOCOM*, 2010.

[62] M. Goetz, J. Leskovec, M. McGlohon, and C. Faloutsos. Modeling blog dynamics. In *Proc. of ICWSM*, 2009.

[63] L. Gorelick, M. Galun, E. Sharon, R. Basri, and A. Brandt. Shape representation and classification using the poisson equation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(12):1991–2005, 2006.

[64] L. Grady and E. Schwartz. Isoperimetric partitioning: a new algorithm for graph partitioning. *SIAM Journal of Scientific Computing*, 27(6):1844–1866, 2006.

[65] D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proc. of ASONAM*, 2010.

[66] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller. Self-similarity in file systems. In *Proc. of SIGMETRICS*, 1998.

[67] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proc. of CIKM*, 2010.

[68] L. Guo, E. Tan, S. Chen, X. Zhang, and Y. Zhao. Analyzing patterns of user content generation in online social networks. In *Proc. of KDD*, 2009.

[69] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization. In *Proc. of SODA*, 2010.

[70] M. Hay, C. Li, G. Miklau, and D. Jensen. Accurate estimation of the degree distribution of private networks. In *Proc. of ICDM*, 2009.

[71] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *PVLDB*, 1(1):102–114, 2008.

[72] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1-2):1021–1032, 2010.

[73] P. Holme and B. Kim. Growing scale-free networks with tunable clustering. *Physical Review E*, 65(2):026107, 2002.

[74] J. Hopcroft and D. Sheldon. Manipulation-resistant reputations using hitting time. *Internet Math*, 5(1-2):71–90, 2008.

[75] J. R. M. Hosking. Fractional differencing. *Biometrika*, 68(1):165–176, 1981.

[76] Z. Huang. Link prediction based on graph topology: The predictive value of the generalized clustering coefficient. In *In Proc. of LinkKDD*, 2006.

[77] G. Jeh and J. Widom. Scaling personalized web search. In *Proc. of WWW*, pages 271–279, 2003.

[78] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. In *Proc. of IMC*, 2010.

[79] S. Kairam, D. Wang, and J. Leskovec. The life and death of online groups: predicting group growth and longevity. In *Proc. of WSDM*, 2012.

[80] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating pagerank computations. In *Proc. of WWW*, 2003.

[81] U. Kang, M. McGlohon, L. Akoglu, and C. Faloutsos. Patterns on the connected components of terabyte-scale graphs. In *Proc. of ICDM*, 2010.

[82] U. Kang, C. Tsourakakis, and C. Faloutsos. Radius plots for mining terabyte scale graphs: Algorithms, patterns, and observations. In *Proc. of SDM*, 2010.

[83] T. Karagiannis, M. Faloutsos, and R. H. Riedi. Long-range dependence: now you see it, now you don't! In *Proc. of IEEE Globecom*, 2002.

[84] D. Karger and M. Ruhl. Find nearest neighbors in growth-restricted metrics. In *Proc. of STOC*, 2002.

[85] N. L. D. Khoa and S. Chawla. Robust outlier detection using commute time and eigenspace embedding. In *Proc. of PAKDD*, 2010.

[86] J. Kim, K. Goh, B. Kahng, and D. Kim. Fractality and self-similarity in scale-free networks. *New Journal of Physics*, 9(6):177, 2007.

[87] M. Kim and J. Han. A particle-and-density based evolutionary clustering method for dynamic networks. *PVLDB*, 2(1):622–633, 2009.

[88] R. Kleinberg. Geographic routing using hyperbolic space. In *Proc. of INFOCOM*, 2007.

[89] S. G. Kobourov and M. Landis. Morphing planar graphs in spherical space. In *Proc. of Graph Drawing*, 2007.

[90] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *Proc. of WWW*, 2005.

[91] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proc. of KDD*, 2006.

[92] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proc. of FOCS*, 2000.

[93] H. Kwak, Y. Choi, Y. Eom, H. Jeong, and S. Moon. Mining communities in networks: a solution for consistency and its evaluation. In *Proc. of IMC*, 2009.

[94] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proc of WWW*, 2010.

[95] P. Lawrence, B. Sergey, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[96] J. R. Lee. Volume distortion for subsets of euclidean spaces: extended abstract. In *Proc. of SCG*, 2006.

[97] M.-J. Lee, J. Lee, J. Y. Park, R. H. Choi, and C.-W. Chung. Qube: A quick algorithm for updating betweenness centrality. In *Proc of WWW*, 2012.

[98] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM TON*, 2(1):1–15, 1994.

[99] W. E. Leland and D. V. Wilson. High time-resolution measurement and analysis of LAN traffic: Implications for lan interconnection. In *Proc. of INFOCOM*, 1991.

[100] J. Leskovec, L. Adamic, and B. Adamic. The dynamics of viral marketing. *ACM TWEB*, 1(1), 2007.

[101] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *Proc. of KDD*, 2008.

[102] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Proc. of PKDD*, 2005.

[103] J. Leskovec and C. Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proc. of ICML*, 2007.

[104] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of KDD*, 2005.

[105] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[106] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proc. of WWW*, 2010.

[107] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. of CIKM*, 2003.

[108] R. Lichtenwalter, J. Lussier, and N. Chawla. New perspectives and methods in link prediction. In *Proc. of KDD*, 2010.

[109] Y. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. Tseng. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proc of WWW*, 2008.

[110] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:577–591, 1994.

[111] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *Proc. of SIGMOD*, 2008.

[112] L. Lovász. Random walks on graphs: A survey. *Bolyai Society Mathematical Studies*, 2:1–46, 1993.

[113] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for internet coordinate systems. In *Proc. of IMC*, 2005.

[114] C. Lumezanu and N. Spring. Measurement manipulation and space selection in network coordinates. In *Proc. of ICDCS*, 2008.

[115] P. Mahadevan, D. Krioukov, K. Fall, and A. Vahdat. Systematic topology analysis and generation using degree correlations. In *Proc. of SIGCOMM*, 2006.

[116] B. B. Mandelbrot and J. W. van Ness. Fractional Brownian motions, fractional noises and applications. *SIAM Review*, 10, 1968.

[117] A. McCallum, X. Wang, and A. Corrada-Emmanuel. Topic and role discovery in social networks with experiments on enron and academic email. *J. Artif. Int. Res.*, 30(1):249–272, 2007.

[118] M. McGlohon, L. Akoglu, and C. Faloutsos. Weighted graphs and disconnected components: patterns and a generator. In *Proc. of KDD*, 2008.

[119] F. McSherry. A uniform approach to accelerated pagerank computation. In *Proc. of WWW*, 2005.

[120] F. McSherry and R. Mahajan. Differentially-private network trace analysis. In *Proc. of SIGCOMM*, 2010.

[121] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proc. of PODS*, 2004.

[122] D. Mir and R. Wright. A differentially private graph estimator. In *Proc. of ICDMW*, 2009.

[123] A. Mislove, K. P. Gummadi, and P. Druschel. Exploiting social networks for internet search. In *Proc. of HotNets*, 2006.

[124] A. Mislove, H. Koppula, K. Gummadi, P. Druschel, and B. Bhattacharjee. Growth of the flickr social network. In *Proc. of WOSN*, 2008.

[125] A. Mislove, M. Marcon, K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proc. of IMC*, 2007.

[126] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. of IEEE Symposium on Security and Privacy*, 2008.

[127] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *Proc. of IEEE Symposium on Security and Privacy*, 2009.

[128] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.

[129] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19:91–113, 2010.

[130] M. Newman. Analysis of weighted networks. *Physical Review E*, 70(5), 2004.

[131] M. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6), 2004.

[132] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2), 2004.

[133] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. of INFOCOM*, 2002.

[134] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. of STOC*, 2007.

[135] R. V. Oliveira, B. Zhang, and L. Zhang. Observing the evolution of internet as topology. In *Proc. of SIGCOMM*, 2007.

[136] G. Palla, A. Barabasi, and T. Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, 2007.

[137] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: A fast and scalable tool for data mining in massive graphs. In *Proc. of KDD*, 2002.

[138] F. Papadopoulos, D. Krioukov, M. Bogu, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *Proc. of INFOCOM*, 2010.

[139] K. Park and W. Willinger. *Self-similar network traffic and performance evaluation*. Wiley Online Library, 2000.

[140] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1995.

[141] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *Proc. of CIKM*, 2009.

[142] K. P. N. Puttaswamy, A. Sala, and B. Y. Zhao. Starclique: Guaranteeing user privacy in social networks against intersection attacks. In *Proc. of CoNEXT*, 2009.

[143] H. Qiu and E. Hancock. Image segmentation using commute times. In *Proc. of BMVC*, 2005.

[144] H. Qiu and E. Hancock. Robust multi-body motion tracking using commute time clustering. *ECCV*, pages 160–173, 2006.

[145] S. Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proc. of SCG*, 1999.

[146] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proc. of SIGMOD*, 2010.

[147] M. J. Rattigan, M. Maier, and D. Jensen. Using of structure indices for efficinet approximation of network properties. In *Proc. of KDD*, 2006.

[148] D. Rybski, S. V. Buldyrev, S. Havlin, F. Liljeros, and H. A. Makse. Scaling laws of human interaction activity. *Proceedings of the National Academy of Sciences*, 106(31):12640–12645, 2009.

[149] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Zhao. Measurement-calibrated graph models for social network experiments. In *Proc of WWW*, 2010.

[150] A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao. Sharing graphs using differentially private graph models. In *Proc. of IMC*, 2011.

[151] P. Sarkar and . W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proc. of UAI*, 2007.

[152] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM ToN*, 12(6):993–1006, 2004.

[153] Y. Shavitt and T. Tankel. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Trans. Netw.*, 16(1):25–36, 2008.

[154] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[155] C. Song, S. Havlin, and H. A. Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005.

[156] H. Song, T. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proc. of IMC*, 2009.

[157] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. of STOC*, 2008.

[158] J. Sun, C. Faloutsos, S. Papadimitriou, and P. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proc. of KDD*, 2007.

[159] G. Swamynathan, C. Wilson, B. Boe, K. Almeroth, and B. Y. Zhao. Do social networks improve e-commerce: a study on social marketplaces. In *Proc. of WOSN*, 2008.

[160] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *Intl. Journal of uncertainty fuzziness and knowledge based systems*, 10(5):557–570, 2002.

[161] M. Takaffoli, F. Sangi, J. Fagnan, and O. Zaıane. A framework for analyzing dynamic social networks. *Applications of Social network Analysis (ASNA)*, 2010.

[162] L. Tang and M. Crovella. Virtual landmarks for the Internet. In *Proc. of IMC*, 2003.

[163] C. Tantipathananandh and T. Berger-Wolf. Constant-factor approximation algorithms for identifying dynamic communities. In *Proc. of KDD*, 2009.

[164] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proc. of KDD*, 2007.

[165] M. S. Taqqu and J. Levy. Using renewal processes to generate long-range dependence and high variability. *Dependence in Probability and Statistics*, pages 73– 89, 1986.

[166] M. S. Taqqu, V. Teverovsky, and W. Willinger. Estimators for long-range dependence: an empirical study. *Fractals*, 3(04):785–798, 1995.

[167] R. Toivonen, J. Onnela, J. Saramaki, J. Hyvonen, and K. Kaski. A model for social networks. *Physica A: Statistical and Theoretical Physics*, 371(2):851–860, 2006.

[168] A. Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Physical Review E*, 67(5):056104, 2003.

[169] D. Veitch and P. Abry. *Wavelet tool.* http://www.cubinlab.ee.unimelb.edu.au/ darryl/research.html.

[170] B. Viswanath, A. Mislove, M. Cha, and K. Gummadi. On the evolution of user interaction in facebook. In *Proc. of WOSN*, 2009.

[171] B. Viswanath and A. Post. An Analysis of Social Network-Based Sybil Defenses. In *Proc. of SIGCOMM*, 2010.

[172] K. Wakita and T. Tsurumi. Finding community structure in mega-scale social networks. *CoRR*, abs/cs/0702048, 2007.

[173] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *Proc. of ICDE*, 2002.

[174] D. Watts and S. Strogatz. Collective dynamics of small-worldnetworks. *Nature*, 393(6684):440–442, 1998.

[175] P. Whittle. Estimation and information in stationary time series. *Arkiv för matematik*, 2(5):423–434, 1953.

[176] W. Willinger, V. Paxson, and M. S. Taqqu. Self-similarity and heavy tails: Structural modeling of network traffic. *A practical guide to heavy tails*, 23:27–53, 1998.

[177] W. Willinger, M. S. Taqqu, W. E. Leland, and D. V. Wilson. Self-similarity in high-speed packet traffic: analysis and modeling of ethernet traffic measurements. *Statistical Science*, 10(1):67–85, 1995.

[178] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM TON*, 5(1):71–86, 1997.

[179] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proc. of EuroSys*, 2009.

[180] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. on Knowledge and Data Engineering*, 23(8), 2010.

[181] L. Yen, D. Vanvyve, F. Wouters, F. Fouss, M. Verleysen, and M. Saerens. Clustering using a random walk based distance measure. In *Proc. of ESANN*, 2005.

[182] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Proc. of IEEE Symposium on Security and Privacy*, pages 3 – 17, 2008.

[183] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *Proc. of SIGCOMM*, 2006.

[184] X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao. On the embeddability of random walk distances. *PVLDB*, 6(14):1690–1701, Sept. 2013.

[185] X. Zhao, A. Sala, C. Wilson, X. Wang, S. Gaito, H. Zheng, and B. Y. Zhao. Multi-scale dynamics in a massive online social network. In *Proc. of IMC*, 2012.

[186] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: Shortest path estimation for large social graphs. In *In Proc. of WOSN*, 2010.

[187] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao. Efficient shortest paths on massive social graphs. In *Proc. of CollaborateCom*, 2011.

[188] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao. Fast and scalable analysis of massive social graphs. *CoRR*, abs/1107.5114, 2011.

[189] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *Proc. of PinKDD*, 2008.

[190] E. Zheleva, H. Sharara, and L. Getoor. Co-evolution of social and affiliation networks. In *Proc. of KDD*, 2009.

[191] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *Proc. of ICDE*, 2008.

[192] M. Zimmer. Facebook data of 1.2 million users from 2005 released: Limited exposure, but very problematic. Blog, February 2011. `http://michaelzimmer.org`.

[193] L. Zou, L. Chen, and M. Özsu. K-automorphism: A general framework for privacy preserving network publication. *PVLDB*, 2(1):946–957, August 2009.